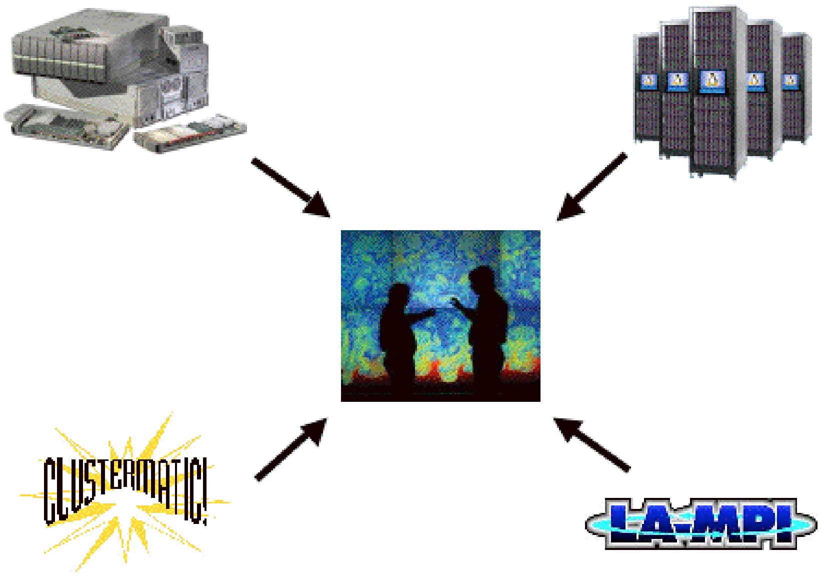
Using the LANL BProc Clusters



Harvey Wasserman

HPC Systems Group (CCN-7)

Los Alamos National Laboratory

http://asci-training.lanl.gov/BProc

May, 2006

hjw@lanl.gov

Last Modified: 05/11/2006 LA-UR-05-2741

Los Alamos National Laboratory
Lab Home | Phone | Search
INSIDE Los Alamos > CCN 7
ICN Consultants
HPC SYSTEMS GROUP
COMPUTING COMMUNICATIONS AND NETWORKING DIVISION

ASCI Training
@LANL
http://asci-training.lanl.gov

Using the LANL BProc Computing Systems

TABLE OF CONTENTS

Course Overview

LANL Computing Strategy

Getting an Account

Clustermatic

- Science Appliance
- Clustermatic Concepts
- Production Clustermatic
- Clustermatic Summary

System Architecture

- Lightning
- Flash
- Pink
- TLC
- Grendels
- Coyote
- Saguaro

Logging In

- Front End Names

Exercise #1

System Configuration Revisited

Exercise #2

Filesystems

File Transfer

Modulefiles

Exercise #3

Submitting Jobs

- Basic Process
- LSF on Lightning
- LSF Queue Structure
- The bpsh Command
- Using MPI
- Exercise #4

Course Overview

- Over the last 2 years Los Alamos National Laboratory has been installing several new supercomputer clusters that run a locally-modified version of the Linux operating system called "Clustermatic." Or sometimes "BProc." Anyway, there now are many of these machines available to users, both ASC-, Institutional Computing, and recharge-funded users. This course is an introduction to these machines.

- BProc systems are easy to use but they are a little different from other Linux systems and from other systems that have been in use at LANL, especially the HP/Compaq Tru64 clusters and machine Lambda.

  Machines in this category include Lightning, Flash, Pink, Grendels, Coyote, Saguaro, and TLC.

- It is assumed that students have familiarity with the LANL computing environment and familiarity with Unix.

  When you finish this course you will:

  - Understand what a Clustermatic system is and how it differs from traditional supercomputer clusters;
  - Understand how the various LANL Clustermatic systems differ from one another;
  - Understand why we are using Clustermatic;
  - Understand how to use these machines to compile, run, and monitor jobs.

Contents

BProc And the LANL Computing Strategy

- All of the Clustermatic systems at LANL essentially serve two purposes:

  1. First and foremost, they are intended to become full production systems, providing high-availability cycles to users on a 24x7 basis supporting local infrastructure and typical LANL computing environments for archival storage, resource management, debugging, network file system, visualization, etc.
  2. They also represent a gateway to future LANL computing models, as shown in the following table.

Monitoring Jobs

- Unix Commands

- LSF Commands

- BProc Commands

Exercise #5

HPSS

- File Transfer Agents

More Hardware Details

- Node & Processor

- 32-Bit Computing

- Interconnect

Porting Considerations

- Endianness

Compiling

- Compile Nodes

- Big File Fix

- Available Compilers

- Compiling MPI codes

- LA-MPI Tips

Exercise #6

Debugging

Exercise #7

Panasas

How to Get Help

Future BProc

Reference Info

Course Evaluation

A New LANL Computing Strategy

Objective	How the Clustermatic Systems Achieve Objective
Better price performance for platforms	Dual-processor nodes with AMD Opteron or Intel Xeon processor and Myrinet interconnect
Leverage open source software wherever possible	64-bit Linux OS LA MPI - more robust and portable message-passing environment than that supplied by hardware vendors
Vendor-independent HPC features needed by ASC applications	Cluster supplied by LinuxNetworX High performance, global, parallel, filesystem provided by Panasas.
Provide a high-availability cluster computing environment	Science Appliance software (LinuxBios, BProc, etc.) provides more efficient large scale cluster system administration. This means more user cycles.

- It is the Science Appliance and BProc software that make Lightning, Flash, Pink, TLC, and Grendels different from other supercomputer clusters. In subsequent sections of this tutorial we explain what this software is and how to use it.

Contents

Getting an Account

LANL USERS: Approval required.

- To get an account on Lightning, use the LANL HPC Accounts web page http://icnn.lanl.gov/accounts/request.php in the red network.
- Note: later in this document we will be discussing a "Lightning" cluster and a "Bolt" cluster; however, for the purposes of accounts, right now these two are the same, so an account on Lightning enables you to use Bolt.
- To get an account on Flash, Grendels, or Sagurao use the LANL HPC Accounts web page http://icnn.lanl.gov/accounts/request.php in the yellow network. Approval is required for both machines. Flash is restricted to users doing NWP work.
- Accounts on Pink, TLC, or Coyote require an Institutional Computing project grant.
    - If you are an IC Principal Investigator you may need to add people to your project. Do this by going to http://icnn.lanl.gov/accounts/admin/addToProject.php. After authenticating with a crypto-card, you will be presented with a list of the projects to which you may add people. On selecting a project an input box will be displayed, as well as a list of the systems upon which the project has an allocation. Enter the Z-numbers of the project members and check boxes next to the system names to create the new accounts.

TRI-LAB USERS and ALLIANCE USERS:

- For the most part, neither Lightning nor Flash are available to off-site users. To apply for an account on Pink use the SNL Sarape Form.

Contents

Software Architecture

What Makes the BProc Systems Different?

- They all use a method of building clusters referred to as a "Science Appliance." Science Appliance actually refers to a redesign of both hardware and software for large-scale clusters. This method was developed by LANL's Advanced Computing Laboratory.

- The main reason for building these machines as Science Appliances is to provide more computing cycles to users.

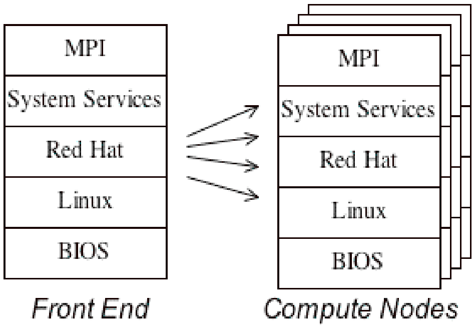
- Overall usability of the cluster is improved by:

    - Reducing hardware and software complexity of the node so that it won't go down as often; and

    - Dramatically reducing the reboot time if the node does go down.

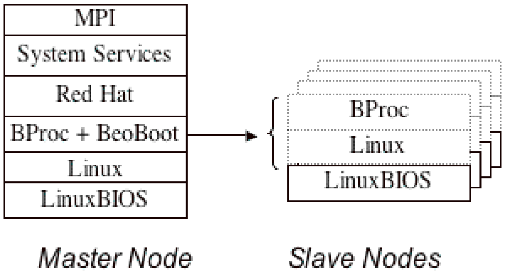
What is a Science Appliance? How Does it Differ from a Traditional Cluster?

- A traditional cluster is built by replicating a complete workstation's software environment on every node.



Traditional Cluster
Architecture

- In the Science Appliance architecture, we have **master nodes** and **slave nodes** and only the master nodes have a fully loaded system. The slave nodes run a minimal software stack.



Science Appliance
Architecture

- The key software in a Science Appliance is an award-winning suite that LANL developed called "Clustermatic". Clustermatic features the Beowulf Distributed Process Space **(BProc)**, LinuxBios, and a variety of other open-source kernel modifications, utilities, and libraries.

- LinuxBIOS and Beoboot are open source products that allow very fast boot times, are remotely accessible, and are designed specifically for cluster systems. For example, the entire Pink cluster can be rebooted in about 7 minutes.

- BProc allows a process space to be shared across multiple nodes in a cluster, even though those nodes run separate system images. Users create processes on the master nodes and the system migrates them (the processes, not the users) to the slave nodes.

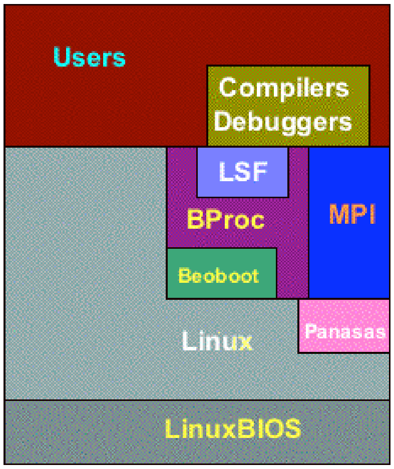
- When a process space is shared this way user processes running on the slave nodes appear as processes running on the master nodes.

- This allows remote process management using the normal UNIX process control facilities (such as **`ps`** and Unix signals) *on the master node*. Standard input, output, and error streams are redirected to the master node.

- One interesting and important fact about BProc systems: The root filesystem is RAM-based.
- More detail on how BProc works is here.

BProc in Production at LANL

- On Lightning, Flash, Pink, TLC, and Grendels a software environment consisting of commonly-used 3rd-party software is layered on top Clustermatic (see figure).



- The software environment on the BProc clusters is still evolving. Furthermore, although all are Linux/BProc systems, there are differences between them, principally due to different computational workloads and user bases. There is an effort to standardize the four systems, though.
- Although the root filesystem in a Science Appliance is RAM-based, the LANL BProc systems allow NFS mounts. The extent to which this happens is the biggest difference between the 4 systems. This, in turn, significantly affects the way you do file input/output.

Science Appliance Summary

- Running jobs on the LANL BProc clusters is easy but is a little different than other LANL systems.
- The system migrates your jobs to the slave nodes but you can follow the jobs' progress from the master node. You can never have a shell on the slave nodes.
- Input/Output from/to the terminal on BProc systems happens just as it does on any other system. Input/Output from/to a file can be different than other systems.
- The slave nodes run a reduced software stack.
- The way you use the front ends is different than other LANL systems.

Contents

System Architecture Overview

- Each cluster is comprised of some number of "nodes" and an interconnect. In all cases, the node is a dual-processor "Evolocity" system from the company Linux Networx.
- (LANL) Terminology: Some of the clusters are partitioned into what we call "segments." The important aspect of segments is that user jobs cannot span them. The segments probably could be quickly combined to create either fewer, larger segments but in practise this hasn't happened in a long time and is relatively unlikely to again.

Lightning:

- April 12, 2005: Total of about 3,060 nodes.
- Lightning contains 13 separate "segments." All segments have 256 nodes, including one master node each. One Lightning segment has dual-core nodes.
- Most segments are used for 32-bit production jobs. Two segments are currently used for 64-bit development. Soon other segments will be converted from 32-bit to 64-bit.
- 2 AMD "Opteron" processors, 1.8, 2.0, or 2.4-GHz, each with 1-MB on-chip Level-2 cache, per node.
- 4-16 GB per node, depending on the segment.
- Includes 128 fileserver nodes not available for computing.

- Classified system for the simulation requirements of the Stockpile Stewardship program. Augments the capabilities of ASC QB and CA/CB/CC.

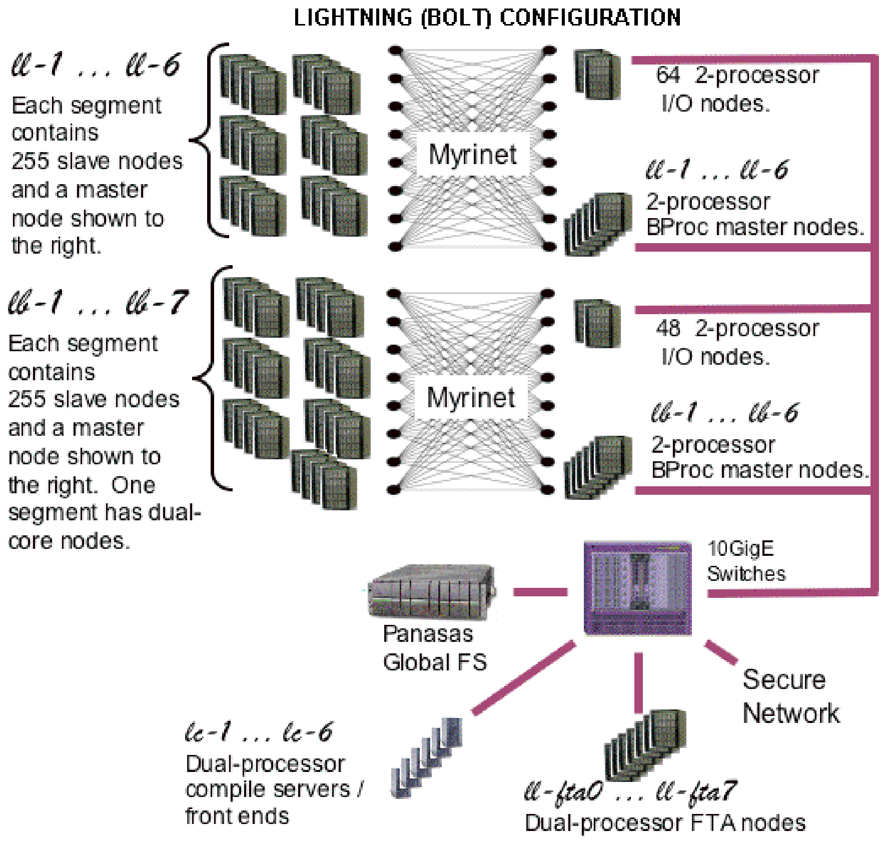
- Used primarily for capacity computing, with a typical capacity job mix of 2-D calculations and possibly smaller 3-D jobs.

- 200 TB Panasas storage.

- Theoretical Peak performance of 30 TeraFLOPS; compare to 10 for ASC QA or QB.

- Seven dual-processor file transfer agents (FTAs).

- A table showing detailed Lightning configuration info is here (LANL Only), although you could get the same info by logging in and using the LSF "lshosts" command.


LIGHTNING (BOLT) CONFIGURATION
ll-1 ... ll-6
Each segment contains 255 slave nodes and a master node shown to the right.
Myrinet
64 2-processor I/O nodes.
ll-1 ... ll-6
2-processor BProc master nodes.
lb-1 ... lb-7
Each segment contains 255 slave nodes and a master node shown to the right. One segment has dual-core nodes.
Myrinet
48 2-processor I/O nodes.
lb-1 ... lb-6
2-processor BProc master nodes.
10GigE Switches
Panasas Global FS
Secure Network
lc-1 ... lc-6
Dual-processor compile servers / front ends
ll-fta0 ... ll-fta7
Dual-processor FTA nodes

Flash

- A 5-segment cluster, between 15 and 300 nodes per segment. Nodes contain 2 Opteron processors at speeds ranging from 2.0 to 2.4 GHz.

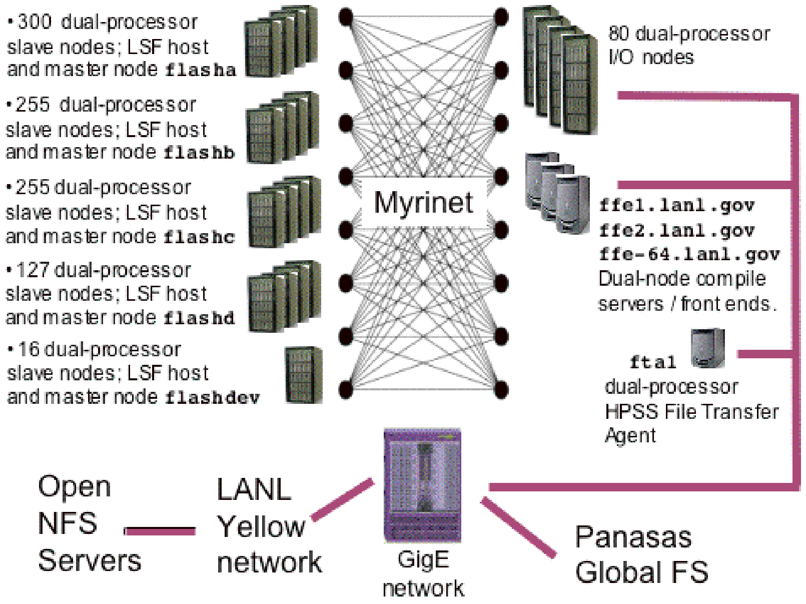
- 80 additional fileserver nodes not available for computing.

- 8-16 GB memory per node.

- Used entirely for unclassified capacity computing - NWP jobs requiring relatively few processors.

- 30 TB Panasas storage.

FLASH CONFIGURATION



•300 dual-processor slave nodes; LSF host and master node **flasha**
•255 dual-processor slave nodes; LSF host and master node **flashb**
•255 dual-processor slave nodes; LSF host and master node **flashc**
•127 dual-processor slave nodes; LSF host and master node **flashd**
•16 dual-processor slave nodes; LSF host and master node **flashdev**
Myrinet
80 dual-processor I/O nodes
ffe1.lanl.gov
ffe2.lanl.gov
ffe-64.lanl.gov
Dual-node compile servers / front ends.
fta1
dual-processor HPSS File Transfer Agent
Open NFS Servers
LANL Yellow network
GigE network
Panasas Global FS

Pink

- A single-segment cluster containing 1,024 nodes.

- 64 fileserver nodes are not available for computing.

- 2 Intel 2.4-GHz "Xeon" processors per node.

- 8KB L1 data cache (2-CP load latency, 64-byte line, one load & one store per CP) & 512-KB Level-2 cache.

- 2 GB memory per node.

- Intel E7500 chipset, 400-MHz system bus.

- Used entirely for Institutional Computing and other non-weapons computing projects.

- Unclassified system on the new TURQUOISE network.

- 32 TB Panasas storage shared with TLC.

TLC

- A single-segment cluster containing 110 user-accessible computing nodes.

- 16 fileserver nodes are not available for computing.

- 2 AMD 2.0-GHz "Opteron" processors, each with 1-MB on-chip Level-2 cache per node.

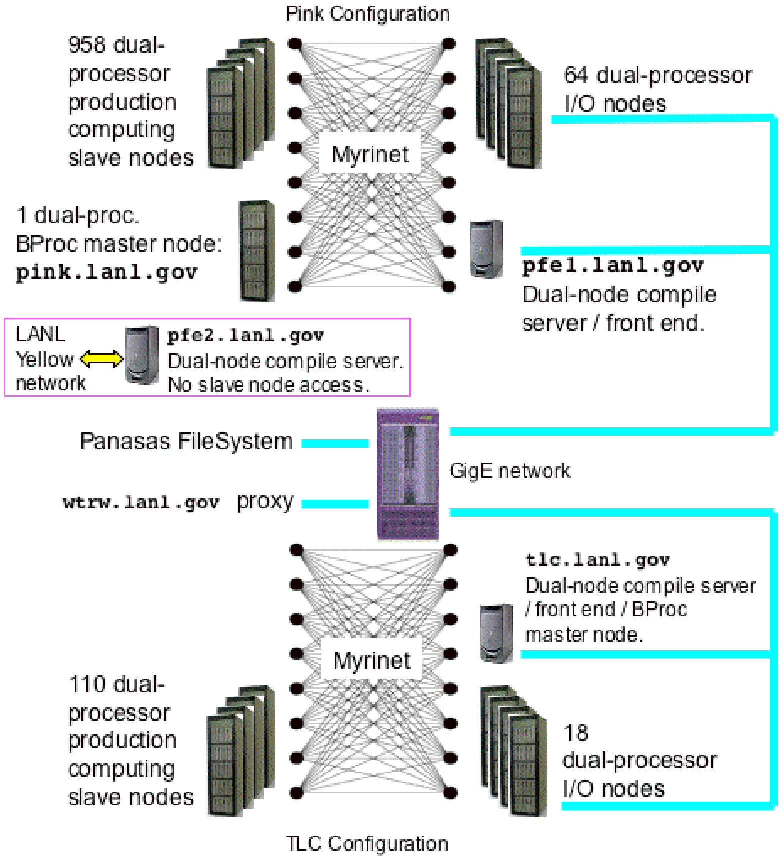
- 8 GB memory per node.

- Disk drive and GigE ports on front end node only.

- Used mostly for Institutional Computing, training, and some code development.

- Unclassified system on the new TURQUOISE network.

- 32 TB Panasas storage shared with Pink.


Pink Configuration
958 dual-processor production computing slave nodes
Myrinet
64 dual-processor I/O nodes
1 dual-proc. BProc master node: pink.lanl.gov
pfe1.lanl.gov Dual-node compile server / front end.
LANL Yellow network
pfe2.lanl.gov Dual-node compile server. No slave node access.
Panasas FileSystem
GigE network
wtrw.lanl.gov proxy
tlc.lanl.gov Dual-node compile server / front end / BProc master node.
Myrinet
110 dual-processor production computing slave nodes
18 dual-processor I/O nodes
TLC Configuration

Grendels

- A single-segment cluster containing 124 user-accessible computing nodes.

- 2 Intel 2.4-GHz "Xeon" processors per node

- 8KB L1 data cache (2-CP load latency, 64-byte line, one load & one store per CP) & 512-KB Level-2 cache.

- 2 GB memory per node

- Myrinet interconnect

- Disk drive and GigE ports on front end node only.

- Used for unclassified capacity computing - NWP M&P codes, Yellow network.

- Currently NO Panasas storage; very limited global storage using NFS.

Saguaro

- A single-segment cluster containing 32 user-accessible computing nodes.

- 2 AMD 2.4-GHz "Operton" processors per node.

- 4 GB memory per node.

- Eternet interconnect.

- NFS-based global temporary storage.

Coyote

- Coyote was acquired through a unique 3-year leasing arrangement with Linux NetWorX,

- 1,406 dual-processor, single-core AMD Opteron nodes.

- 14 TeraFlops total peak performance.

- 11 TeraBytes total memory.

- Shares 160 TB global disk storage with other IC resources.

- Shares 2000 TB archival storage with other IC resources.

- Hierarchical InfiniBand interconnect.

- 64-bit FC3 Linux + BProc V4 operating system with 2.6.14 kernel.

- 10 additional nodes set aside for sequential computing.

- Mellanox AuCD 2.0 - OpenSM/Gen2

- Coyote has a loosely-coupled cluster architecture. There are six "tier-I" clusters (or "segments"), five of which (CY-1, CY-2, ... CY-5) have 272 nodes, while the remaining one (DOT-X) has has 42 nodes.

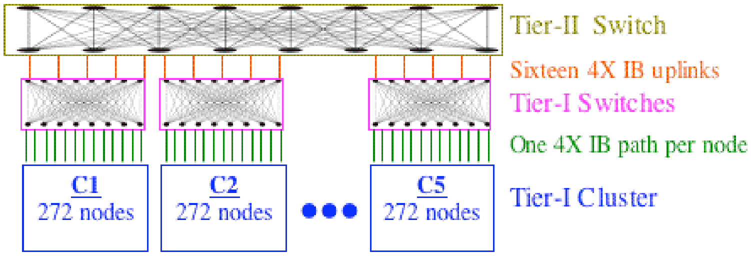
- Nodes within a tier-I system are interconnected via a tier-I 288-port PCI-Express-connected Voltaire 4X InfiniBand switch. There is no federation in the network. It is possible that in the future two tier-I segments may be combined into a single segment by "cross-connecting" through one of the 288-port switches.

- The five big Tier-1 systems all have one BProc master node and 258 BProc compute nodes. They also have 13 I/O nodes not accessible by users. The CY-X system has one BProc master node and 36(?) BProc compute nodes, plus 4 I/O nodes.

- In each Tier-I segment one of the 13 I/O nodes is a master I/O node that manages the process space via BProc for the other 12 I/O nodes. The I/O masters each have one GigE connection to the external Turquoise network and one connection to an internal Coyote hardware monitoring network. The I/O master nodes also provide OpenSM (subnet manager) services for each Tier-I segment. Each I/O compute node has one IB connection to the Tier-I IB switch.

- All nodes have 8GB PC3200 registered ECC memory.

- Coyote also has associated with it 10 nodes for serial processing (cy-s1, cy-s2, ... cy-s10).



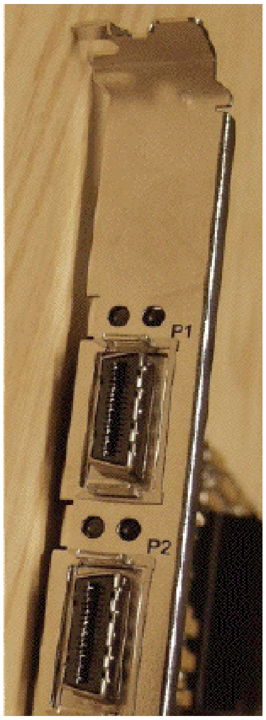
- The most important difference between Coyote and the SGI Altix system used in LANL's Mauve supercomputer is that Coyote is basically a "distributed memory" supercomputer. Each of the Coyote nodes will be running its own operating system and codes cannot "share" memory beyond a single node. This also means that codes will be restricted to the physical memory available on a single node; they cannot allocate memory on "distant" nodes, as was possible on Mauve.

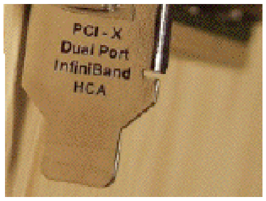
Comparison Of Lightning and C Cluster Segments	
Lightning	**CA, CB, or CC**
255 compute nodes per segment	126 compute nodes per segment
510 compute processors per segment	504 compute processors per segment
4.0 GigaFlops peak per processor	2.5 GigaFlops peak per processor
2.0 TeraFlops peak per segment	1.3 TeraFlops peak per segment
8 GB RAM per node (4 GB per processor)	4 GB RAM per node (1 GB per processor)

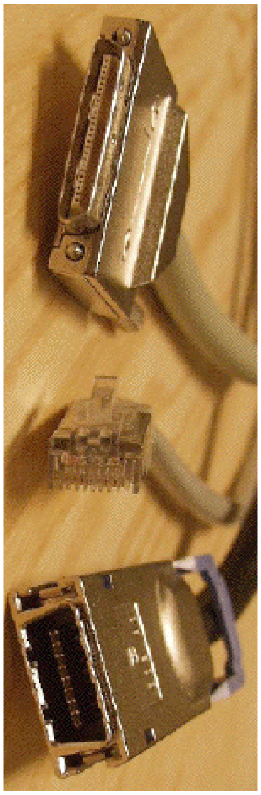
Contents

Logging In

- The BProc systems all have front ends, which are the **ssh** gateways but are also used for other tasks (such as compiling). In this regard they are different from other LANL systems (such as QB, QSC, etc.). See the Compiling section below.
- It's important for you to understand that the front ends are different from the BProc master nodes, except on TLC. They are different hardware and the terms "front end" and "master node" are not synonymous.
- Another key point: Because Lightning and Flash are in the midst of a transition from 32-bit to 64-bit computing, some of their front ends may be different in terms of compilation but the front ends are still identical in terms of accessing the slave node segments.
- There are seven (7) Lightning front ends, called ***lc-1***, ***lc-2***, ***lc-3***, ***lc-4***, ***lc-5***, ***lc-6***, and ***lc-64***. These front ends are the gateway to all 13 Lightning segments - you can use any one of them. However, ***lc-6*** and ***lc-64*** are 64-bit systems; the others are 32-bit systems.
- Flash has three front ends (ffe1, ffe2, and ffe-64) that are equivalent in terms of compute node access; however, ffe1 and ffe2 are 32-bit systems and ffe-64 is a 64-bit system. See the 64-bit computing section.
- Pink has two front ends, pfe1 and pfe2. One of these, pfe1, is in the LANL Turquoise network. The other, pfe2, is in the LANL Yellow network!
- TLC has one front end, tlc.lanl.gov, which is in the Turquoise network.
- Grendels has one front end, gfe1.lanl.gov, which is in the Yellow network.
- The four Coyote front ends are cy-c1, cy-c2, cy-c3, cy-c4. They are all the same in terms of reaching the Coyote segments and BProc master nodes, i.e., all four "see" the entire Coyote cluster.
- Saguaro has one front end. Remember - this is not the master node.
- To log in to a Red or Yellow network machine, use **ssh** to one of the front end systems listed in the table below.
- To log in to a Turquoise network machine, two steps are required. First, use **ssh** to Turqoise proxy **wtrw.lanl.gov** and then use **ssh** one of the front end systems listed in the table below. These two steps can be combined, as follows:
  ssh -t wtrw.lanl.gov ssh pfe1 .







BProc System Front Ends			
System		Front End Names	Segment / LSF Host / Master Node Names
Secure	Lightning	*lc-1, lc-2, lc-3, lc-4, lc-5, lc-6*	*ll-1, ll-2, ll-3, ll-4, ll-5, ll-6; lb-1, lb,-3, lb-4, lb-5, lb-6, lb-7*
Yellow	Flash	ffe1, ffe2, and ffe3	flasha, flashb, flashc, flashd, flashdev
	Grendels	gfe1	grendels
	Saguaro	sfe	saguaro
Turquoise	Pink	pfe1 and pfe2	pink
	TLC	tlc	tlc
	Coyote	cy-c1, cy-c2, cy-c3, cy-c4	cy-1, cy-2, cy-3, cy-4, cy-5

Exercise

Exercise #1: Logging In

1. **Log in to one of the workshop machines**. This will be either Flash, Pink, or TLC. Use **ssh** and authenticate with your CryptoCard.

2. Obviously, there isn't too much we can do yet because we haven't learned how to run jobs or compile. Try running the **top** command. If you've never used it before, try **man top**. Later, we will learn about another kind of **top** command that is important for BProc systems.

3. **Watch Your "Dot!"**

   Type

   ```
   echo $PATH
   ```

   Look at the result and make sure that a period appears before, in between, or after a colon (:). Here is an example of how it might appear:

   ```
   /usr/kerberos/bin:/usr/local/bin:/bin:/lsf/bin:/usr/X11R6/bin:.
   ```

   In this example, it appears at the end. If the period does not appear somewhere in your path, then type

   ```
   set path=($path .)
   ```

   Then type **echo $PATH** again to make sure your change worked.

   It is a very good idea to amend one of your "dot" files (.login or .cshrc or something similar) to include a line that adds dot to your path. Caution: when you edit your dot files you should always have two windows open on the machine - one for editing and one for testing.

   The shell path variable tells the system where to look for programs that you might want to run. Having "dot" in your path tells the system to look in the current directory so that when you execute a file (e.g. **a.out**) you can just type **a.out** instead of **./a.out** .

This ends the first exercise.

Exercise #2: PS on BProc

1. In this exercise we will very briefly look at the Unix **ps** command on BProc systems. On any Unix system the **ps** command gives a snapshot of all current processes.

2. Make sure you are logged into a BProc front end system.
3. Type the following command. Your best bet is to copy and paste it using the mouse.

   ```
   ps -elf | sed '/root/d'|more
   ```

4. Observe the output briefly. You will see a lot of system-related processes.
5. Observe your prompt. Then type **`llogin`**. When this command finishes you will be on a master node. Observe your prompt again. Which segment of the machine are you on?
6. Then input that same **`ps -elf | sed '/root/d'|more`** command. What do you observe that's different?
7. You should see a lot of user processes where the command name is enclosed in **[**square brackets**]**. Those are ghost processes representing processes that are actually running on slave nodes. If you don't see any of these, perhaps no one is running.
8. Log out of the master node by typing "exit."

This ends the second exercise.

Contents

Filesystems

- This is an area where there are significant differences between the BProc clusters and between the BProc and other LANL clusters.
- Special **Security Note**: The Turquoise network is designed to enhance collaboration between Los Alamos and external scientific institutions. No export-controlled code or data is allowed in the Turquoise Network. You may compile export-controlled source code on pfe2.lanl.gov in the Yellow network and transfer the binary to a Turquoise filesysem to run the code.
- The following filesystems are available on Pink, TLC, and Coyote:

- Your *home* directory.
- Your *scratch* directories.
- The *project* directories.
- The *archive* directories.

- The scratch, project, and archive directories are the same on all Turquoise network systems. The home directories are not!
- HPSS does not exist on the Turquoise network; hence, Pink, TLC, and Coyote do not have direct access to it. You must store or retrieve files to HPSS in the yellow network and transfer them to the Turquoise separately.

Your Home Directory

- In the Turquoise network, the good news is that you have one; the bad news is that you can't do much with it.
- On Lightning, Flash, and Saguaro home directories are cross-mounted with other systems in their respective networks - i.e., they are identical.
- In the Turquoise network home directories exist only on local disk space on the front ends. These should contain only dot files, although the .login file can be set up to move the user directly to a cross-mounted working space. This differs from the user home space location in the Yellow Network.
- Per-user space in $HOME on the Turquoise network is limited to 50MB. This is tiny! **Remember**: If your $HOME space fills up you will not be able to use LSF but the error message you get will be obscure. Home directories are not designed for daily usage or working space.

  A full home space can also cause problems with X clients and again, the error message will be obscure. Always remember to check for files with **`ls -al`** since there may be many beginning with a period that won't show up with **`ls`**.

- Repeat: Home directories are not available on the Pink, TLC, or Coyote master nodes or on the Pink, TLC, or Coyote compute nodes.

The Projects Directories

- In the Turquoise network a new directory called **/usr/projects** has been created, cross-mounted on all Turquose machines. Under **/usr/projects**, there are directories for each project that has an Institutional Computing resource allocation. Under **/usr/projects/<*your_project*>** you will find a directory for yourself (your moniker), as in **/usr/projects/<*your_project*>/<*your_moniker*>**. You can use this area for storage.

  Example: If your project is named W04_plasma the new directory will be called plasma.

- To find which group you're in use the LSF **bugroup** command and **grep** for your moniker or user id from the output.
- You need to **cd** to the **/usr/projects/<*your_project*>** directory for the automounter to mount it. If you just use **ls**, you might not see it. If you can't find your new working directory, send email to consult@lanl.gov.
- *On Pink, TLC, and Coyote the $HOME and /usr/projects filesystems are not available on the compute nodes.* Panasas is the only filesystem available on the compute nodes on these systems.
- On Lightning and Flash NFS-mounted project spaces are mounted on all front ends, master nodes, and slave nodes. On Lightnint the **/netscratch** filesystem is also mounted on all front ends, master nodes, and slave nodes.

Your Scratch Directories

- **Panasas**. Temporary ("scratch") storage on Lightning, Flash, Pink, TLC, and Coyote is via the Panasas storage cluster and PanFS parallel filesystem, which is mounted as

  /net/scratch1

  and

  /net/scratch2

  and is common to all three Turquoise network clusters. These globally accessible filesystems are on all front ends, BProc master nodes, and BProc compute nodes. This is where you want to do your runs.

- The Panasas file spaces ARE NOT BACKED UP.
- Currently, there are no user-level quotas on Panasas. However, all of the Panasas file systems have a hard quota that prevents any further writing when 95% capacity is reached.
- Panasas is NOT on Saguaro. Instead, there is an NFS-mounted filesystem for temporary data called ...*we don't know yet.* It might be **/scratch**. It might be **/net/scratch** or **scratch1**. The important difference from Lambda is that it will be globally named, meaning it will have the same name from the front end, from the master node, and from all compute nodes!
- **Panasas on Lightning**. NOTE: BIG CHANGES COMING HERE. READ CAREFULLY. Temporary ("scratch") storage on Lightning is via the Panasas storage system and PanFS filesystem. The mount point for these is changing and this requires action from current users.

  The mount points for these on Lightning are **/net/scratch1**

  and

  /net/scratch2

  but only until *until Wednesday, April 5.* On that date these will go to read-only. Two weeks after that, these file systems will be unavailable.

  On Lightning a new file system, called **/scratch3**, is available now with 87TB. Note the change in naming convention (done to improve consistency between various LANL clusters).

  You are required to migrate your data from the Lightning's **/net/scratch1** and **/net/scratch2** file systems to **/scratch3** as we plan to combine these into one big scratch file system. All user data will be DESTROYED in the /net/scratchX areas.

- All of the Panasas file systems have a hard quota that prevents any further writing when 95% capacity is reached. there are currently no user-level limits.
- **NOTICE!!!!** File purging for the Panasas file systems on Lightning will begin on February 13, 2006. The purge policy will be the same as what we have currently for the Q's/C's. You can view that policy on http://computing.lanl.gov/article/161.html.

The Archive Directories

- The Turquoise network has a new archive space via the NFS-mounted Tivoli Storage Manager (TSM). The path is **/archive/*your_project*/*your_moniker***. This is available on all Turquoise systems, master and front ends only. To request space contact Tom Stup, whose e-mail is tds.
- In this space there is a 20-TB disk cache backed up by a 2-PB tape archive.

Saguaro and Grendels Filesystems

- Grendels: Yellow-network NFS-mounted $HOME and project spaces are NOT mounted anywhere. Grendels has its own, locally-mounted $HOME and scratch space (**/scratch1**).
- Saguaro: Yellow-network NFS-mounted $HOME and project spaces are available. Additionally, Saguaro has its own temporary scratch space, NFS-mounted as **/scratch1**.

Local BProc Filesystems

- Each node has a local file space that is entirely RAM based. The only filesystem mounted there is **/tmp**.
- I/O would be very fast in this space. However, there are three enormous problems:
  - If this space fills, the node will crash. Remember that shared libraries exist in this space, as does the OS.
  - The front ends and master nodes cannot see this space. Special BProc commands have to be used to copy files to and from it.
  - This space is wiped when the node is rebooted.

Filesystem Summary

- Available filesystems are summarized in the table below ("FE" means filesystem is mounted on the front end; M = mounted on the master node; S = slave nodes).

Available Filesystems						
Filesystem	**Lightning**	**Flash**	**Pink**	**TLC**	**Grendels**	**Saguaro**
NFS Home Directory	FE, M, S	FE, M, S	pfe2 only	-	-	FE, M, S not sure
NFS /usr/projects	FE, M, S	FE, M, S	-	-	-	FE, M, S not sure
NFS /netscratch	FE, M, S	-	-	-	-	-
PanFS /net/scratch1 & /net/scratch2	FE, M, S	FE, M, S	FE, M, S	FE/M, S	-	-
Local Home Directory	-	-	pfe1 & pink only	FE/M	FE, M, S	-
Turquoise /usr/projects	-	-	pfe1 & pink only	FE/M	-	-
/scratch1	-	-	-	-	FE, M, S	FE, M, S
/archive	-	-	FE, M	FE/M	-	-

Contents

File Transfer

- For Lightning, Flash, and Grendels use scp to transfer code/data between these clusters and other systems on their respective networks.
- The **give** command is on Lightning and Flash.
  - The recipient should copy the file from

    /net/givedir/*userid*.
  - The destination directory is global across all Lightning segments and front ends but can't be seen from any of the other LANL clusters.
  - **give** is not on grendels, pink, or TLC.
- To transfer a file from the Yellow network to Mauve, Pink, or TLC you need a "two-hop" **scp**. Examples of transfers in both directions are shown below. Transfers from Turquoise to the Yellow must be initiated from the Yellow network.

scp Between Yellow and Turquoise Networks

```
qscfe1% scp hjw@wtrw.lanl.gov:mauve:/scratch/hjw/file1 .
hjw@wtrw.lanl.gov's password:
file1   100% |********************************|  2900       00:00

qscfe2%  scp file2 wtrw.lanl.gov:pfe1.lanl.gov:/net/scratch/hjw
hjw@wtrw.lanl.gov's password:
file2   100%   33KB   1.8MB/s   00:00
```

File Transfer Agents (FTAs)

- File Transfer Agents are special hardware units for transferring files. Their purpose -- and the way you use them -- differs amongst the various networks.
- On Lightning use FTAs via LSF to transfer files to/from HPSS. See the HPSS section below for more info.
- On Turquoise network machines use FTAs via scp, sftp, or rsync to transfer files to/from yellow network machines.

Turquoise Network File Transfer Agents

- For data in the Turquoise **/usr/projects** spaces or on PanFS, the file transfer agents (FTA) tetsuo and akira are available for direct **sftp** or **scp**. More info is available on the Turquoise web page. Here are examples of the usage of tetsuo.lanl.gov:

scp Between Yellow and Turquoise Networks

```
qscfe1% scp tetsuo.lanl.gov:/usr/projects/support/hjw/data .
hjw@tetsuo.lanl.gov's password:
data                           100% 3844   110.8KB/s   00:00

qscfe1% sftp tetsuo.lanl.gov
Connecting to tetsuo.lanl.gov...
hjw@tetsuo.lanl.gov's password:
sftp> cd /net/scratch1/hjw
sftp> ls
.
..
1GB
3GB
datafile
```

```
fakedata
tmp
```

Contents

The Module Command and Modulefiles

- In this section we assume that you know how to use the module utility and modulefiles. If you don't, read the introduction on computing.lanl.gov.
- There are important differences in the way modules are implemented on Lightning, Flash, and Pink - they are very different from other LANL systems and they are still evolving on the BProc systems.
- Modulefile names have descriptive prefixes on these systems. The prefixes are package based, such as "Intel," "totalview," and "lampi."
- Where a package has more than one modulefile available, one will generally be designated as "(default)." You can use this modulefile by just giving its package name. Example below.
- If a package does not have a default (true only for "Intel") you can still specify the modulefile giving just the package name; however, the modulefile you'll get is the one with the lowest alphanumeric string.
- There are also modulefile "groups" (compiler, mpi, debugger, tools, misc). When you list available modulefiles the groups are set off from one-another, and the group name is always shown on the left. The groups are also listed alphabetically.
- You cannot load conflicting modulefiles, such as two MPI packages or two Fortran compilers. An error will result. If you want to override this you can, by setting an environment variable BUT YOU NEED TO BE CAREFUL -- OKAY YOU HAVE BEEN WARNED!!!!:

  ```
  setenv IGNOREMODULECONFLICTS 1
  ```

  Be careful if you expect environment variables such as **CC** to be set.
- A new `module` command is `module help <Group_name>`, which simply lists all modulefiles available for a given group.
- Here are some examples:

Modulefile Usage on BProc Systems

```
lc-1% module avail
-------------- /usr/share/modules/modulefiles/compiler -----------------
Compilers:        intel/8.1-fortran    nag/4.2-x86      pgi/5.2-4(default)
absoft/8.0        lahey/6.1e(default)  nag/5.0-x86(default)
intel/7.1         lahey/6.2            pgi/5.0-2
intel/8.1-c       nag/4.2-amd64        pgi/5.1

--------------- /usr/share/modules/modulefiles/debugger ----------------
Debuggers:              totalview/6.5.0-2(default)
totalview/6.4.0-2       totalview/6.6.0

----------------- /usr/share/modules/modulefiles/misc ------------------
Misc:        module-info modules     use.own

----------------- /usr/share/modules/modulefiles/mpi -------------------
MPI_Libraries:         lampi/1.5.8           mpich/1.2.5
lampi/1.5.10(default) lampi/1.5.9

---------------- /usr/share/modules/modulefiles/tools ------------------
Tools:         flint/5.00.20             purify/6.0
hdf5/1.6.1     procmon/2.0.2
ups/2.7.3      valgrind/2.2.0            vampir/3.5.0-lampi(default)
java2sdk/1.4.2
```

```
lc-1% module load lahey
lc-1% module list
Currently loaded Modulefiles:
  1) lahey/6.1e
lc-1% module avail lampi
lampi/1.5.10(default) lampi/1.5.8           lampi/1.5.9
lc-1% module load lampi
lc-1% module list
Currently loaded Modulefiles:
  1) modules        2) lahey/6.1e     3) lampi/1.5.10
lc-1% module load intel/8.1-c
lc-1% module load intel/8.1-fortran
ERROR: Module 'intel/8.1-fortran' conflicts with a currently
   loaded module 'lahey/6.1e'
```

- The **module** command has the following (typical) options.

module avail	List all available modulefiles.
module list	List all loaded modulefiles.
module load *modulefile [modulefile ...]*	Load modulefile[s] into the current shell environment.
module unload *modulefile [modulefile ...]*	Unload modulefile[s] from the current shell environment.
module switch *modulefile1 modulefile2*	Unload modulefile1 and load modulefile2.
module show *modulefile*	Display information about a modulefile.
module help *modulefile*	Display information about a modulefile.
module whatis *modulefile*	Display information about a modulefile.

- You don't need to initialize the modules environment; the system default is to permit their use.
- On all BProc systems you can use the module command on the front ends. (Different from how other LANL systems work.)

  This means you can use them before **llogin**.
- More info on using modules in scripts later, after we've learned how to run jobs.
- As with other LANL system, error messages associated with modulefile problems can be obscure. The following example error message results from not preloading an MPI modulefile:

Sample Error From Not Loading MPI Modulefile

```
ll-2%  mpirun -np 4 sweep3d.mpi
One of --gm or --p4 is required.
```

- Another good one is "Not enough nodes to allocate all processes."
- Also, some of the compilers require that you load their modulefile just to run (not to compile or link). An example of an error message you'd get (from the Intel compiler) if you forget this is:

Sample Error Message From Not Loading the Intel Compiler Modulefile

```
a.out: error while loading shared libraries: libimf.so:
cannot open shared object file: No such file or directory
```

Contents

Exercise

Exercise #3: Working With Modulefiles and Filesystems

1. Type the command **`module avail`**. Then try to figure out what will be the result of "**`module load compiler`**. What will be the result of " **`module load intel`**?" See if you are correct.

2. Determine how much space is available in both of the Panasas scratch filesystems. Write your answers here:

Filesystem	Size	Used	Avail	Percent Used
`/net/scratch1`				
`/net/scratch2`				

3. Make sure you have a directory in both of the scratch filesystems. Type

   `cd /net/scratch1`   and then   **`ls -al`**

   and look for your user id. Verify that the permissions are as you want them. Then do the same for **`/net/scratch2`**.

This ends the third exercise.

Submitting Jobs

Basic Job Submit Process

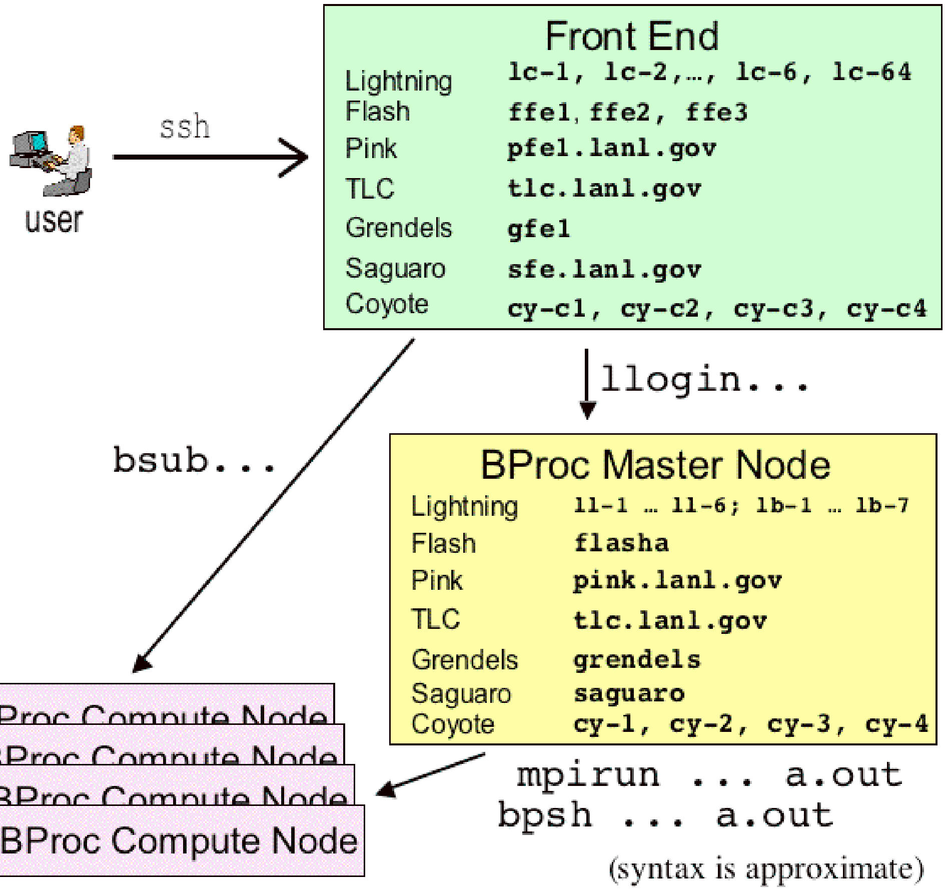
- LSF is on Lightning, Flash, Pink, Grendels, and TLC, and you must use it to submit jobs to run on the slave nodes. As with other LANL systems, you can submit both interactive and batch jobs.

- If you don't use LSF your job will run on the front end nodes. You don't want to do this!!
  - This is different from some other LANL systems (e.g., Q) where you can't execute on the front end generally.

- Terminology: On BProc machines users never "log in" to the slave nodes; you're never "on" the slave nodes. However, you still have to use LSF **`llogin`** to run interactively!

- Using LSF is a necessary but not a sufficient condition. The job submission process potentially involves a combination of LSF and a BProc command. The LSF commands are the same as on other LANL systems; the BProc command will be covered in detail below.

- The basic process is, you first use LSF to obtain an allocation of slave node processors; then when you run your program BProc will migrate it to the processors LSF allocated to you.
  - Interactive use appears a little bit different than on other LANL systems. If you run interactively using **`llogin`**, (or, equivalently, use **`bsub -Is ...`**) you are allocated slave node processors by LSF but your shell will be on the master node. On BProc systems you can never have a shell on a slave-node system.
  - Of course, when **`llogin`** runs it starts a *new* shell on the master node.
  - Note that on Lightning, Flash, and Pink the front end is different from the master node(s). However, currently, on TLC, the front end is the same as the master node. So when you use **`llogin`** on TLC, LSF allocates slave node processors for you and gives you a *new* shell on the master node.



- The syntax in the figure above is approximate; other **llogin**, **bpsh**, **bsub**, and/or **mpirun** options may be used.

- Note: Currently, OpenMP and pthread codes do not work on any of the 32-bit BProc systems. This is a kernel limitation that is removed in the 64-bit systems

- Two examples of interactive use are shown below for Lightning and TLC. The user has used **ssh** to log in to the front ends, either ***lc-1*** or tlc.lanl.gov.

  She then submits **llogin** to the default interactive queue requesting 6 processors. Three nodes are allocated to her, on the LSF host ***ll-2*** in the first case, and on tlc in the second. Her login shell starts on the master node associated with these, which in the Lightning case is a different machine than from where she issued **llogin** but in the TLC case is the same as where she issued **llogin**.

  Now she can run interactively using the nodes 4, 11, and 12 on Lightning and 1 through 3 on TLC. If she wants to use all three she'll run a code using **mpirun** `....` If she wants to use just one node she'll run a code using **bpsh** `....`

Obtaining a Slave Node Allocation Using llogin: Lightning

```
she@lc-1%  llogin -n 6
Job <112> is submitted to default queue <lightq>.
<<Waiting for dispatch ...>>
<<Starting on ll-2>>
NODES: 4,11,12

she@ll-2% bjobs
JOBID  USER  STAT QUEUE   FROM_HOST EXEC_HOST JOB_NAME SUBMIT_TIME
942    she   RUN  smallq  lc-1      6*ll-2    llogin   Jan 29 10:44

she@ll-2%  hostname
ll-2.lanl.gov

she@ll-2%   env|grep NODE
NODES=4,11,12
NODELIST=4,4,11,11,12,12
```

Obtaining a Slave Node Allocation Using llogin: TLC

```
she@tlc%  hostname
tlc.lanl.gov

she@tlc%  llogin -n 6
Job <112> is submitted to default queue <devq>.
<<Waiting for dispatch ...>>
<<Starting on tlc>>
NODES: 1-3

she@tlc% bjobs
JOBID  USER  STAT QUEUE   FROM_HOST EXEC_HOST JOB_NAME SUBMIT_TIME
942    she   RUN  devq    tlc       6*tlc     llogin   Jan 29 10:44

she@tlc%  hostname
tlc.lanl.gov

she@tlc%   echo $NODES
1,2,3
```

X Authorities on Pink

- Special note about X from Meghan Quist of the ICN Consulting Office: Because home directories are not shared between the Pink front end pfe1 and the pink master node, when you **ssh** to pfe1 an .Xauthority file with DISPLAY information is made but when you **llogin** to pink (see below) you don't get the new entry. So, for now, you must manually add it. Here's how:

Setting X Display on Pink

```
pfe1%   env | grep -i disp

DISPLAY=pfe1.lanl.gov:24.0

pfe1%  xauth list
pfe1.lanl.gov:24 MIT-MAGIC-COOKIE-1 259c48a166888d56918c111c11111c1c

pfe1%  llogin
< snip - llogin stuff not shown >

pink%  xclock
X11 connection rejected because of wrong authentication.
X connection to pfe1.lanl.gov:13.0 broken (explicit kill or
server shutdown).
```

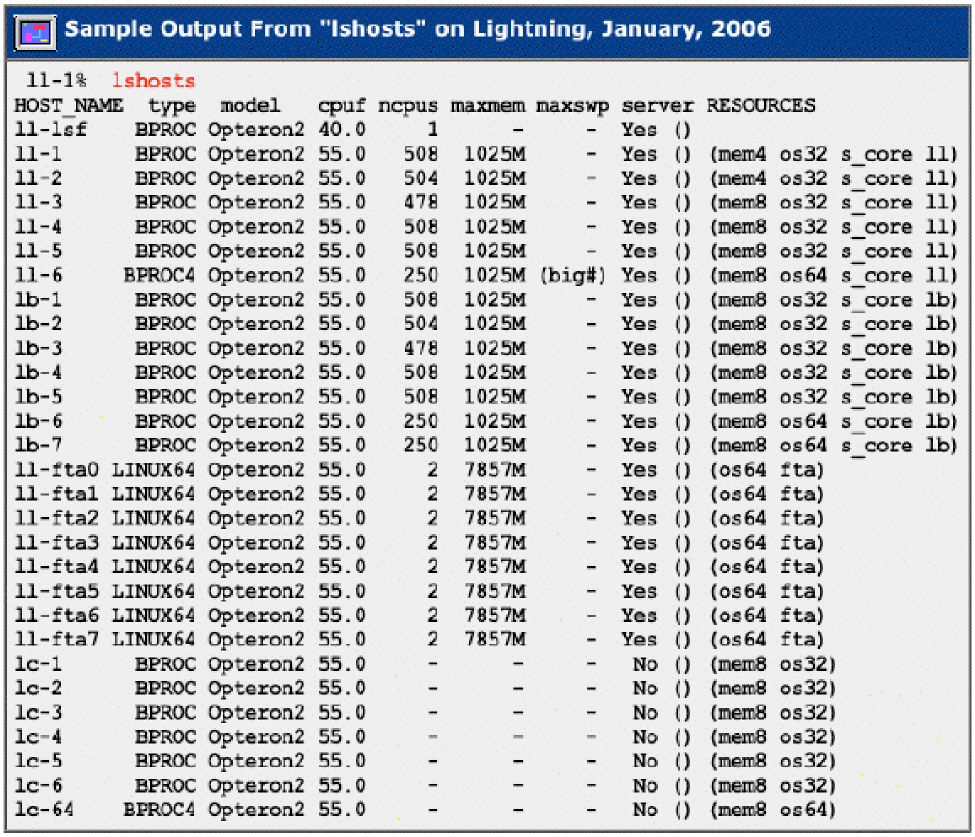
```
pink% xauth add pfe1.lanl.gov:24 MIT-MAGIC-COOKIE-1
259c48a166888d56918c111c11111c1c
pink%  xclock    (This WORKS!)
```

LSF on BProc Systems

- The front-end systems for the BProc clusters (*lc-1* through *lc-64*, ffe1, ffe2, ffe-64, pfe1, tlc, gfe1, and sfe) are LSF submit hosts. This means that you can run all LSF commands from these systems; i.e., you can submit jobs from there with **`bsub`**; you can **`llogin`** from there; and you can monitor job progress using **`bjobs`** from there.

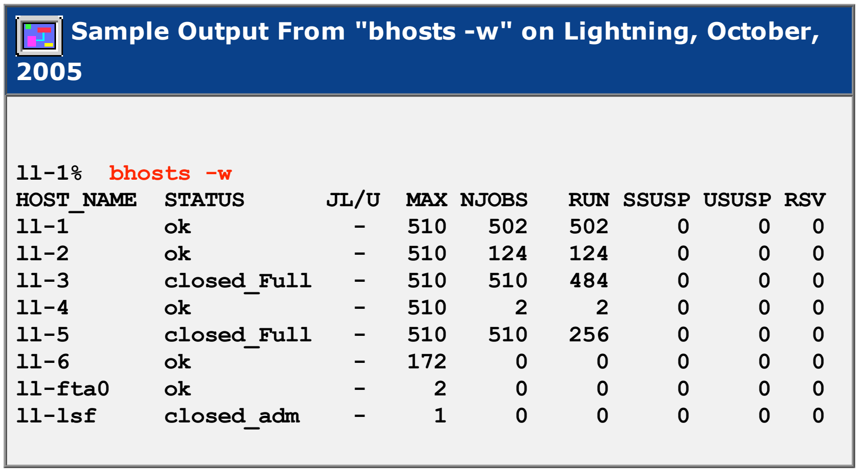
  Which segment of the multisegment systems (Lightning, Flash, Coyote) your job lands on depends on the LSF queue structure. Note that this is consistent with the way LSF works on all LANL clusters. Also, remember: no cross-segment jobs.

  (*Currently*, the Lightning systems (*ll-1, ll-2, ll-3, ll-4, ll-5,* and *ll-6*) and (*lb-1, lb-3, lb-4, lb-5,* and *lb-7*) are simultaneously front ends, LSF hosts, and BProc master nodes. This is expected to change soon, after which, they will no longer be front ends.)



Sample Output From "lshosts" on Lightning, January, 2006

```
 ll-1%  lshosts
HOST_NAME   type   model    cpuf ncpus maxmem maxswp server RESOURCES
ll-lsf      BPROC Opteron2 40.0     1      -      -  Yes  ()
ll-1        BPROC Opteron2 55.0   508  1025M      -  Yes  () (mem4 os32 s_core ll)
ll-2        BPROC Opteron2 55.0   504  1025M      -  Yes  () (mem4 os32 s_core ll)
ll-3        BPROC Opteron2 55.0   478  1025M      -  Yes  () (mem8 os32 s_core ll)
ll-4        BPROC Opteron2 55.0   508  1025M      -  Yes  () (mem8 os32 s_core ll)
ll-5        BPROC Opteron2 55.0   508  1025M      -  Yes  () (mem8 os32 s_core ll)
ll-6       BPROC4 Opteron2 55.0   250  1025M (big#)  Yes  () (mem8 os64 s_core ll)
lb-1        BPROC Opteron2 55.0   508  1025M      -  Yes  () (mem8 os32 s_core lb)
lb-2        BPROC Opteron2 55.0   504  1025M      -  Yes  () (mem8 os32 s_core lb)
lb-3        BPROC Opteron2 55.0   478  1025M      -  Yes  () (mem8 os32 s_core lb)
lb-4        BPROC Opteron2 55.0   508  1025M      -  Yes  () (mem8 os32 s_core lb)
lb-5        BPROC Opteron2 55.0   508  1025M      -  Yes  () (mem8 os32 s_core lb)
lb-6        BPROC Opteron2 55.0   250  1025M      -  Yes  () (mem8 os64 s_core lb)
lb-7        BPROC Opteron2 55.0   250  1025M      -  Yes  () (mem8 os64 s_core lb)
ll-fta0 LINUX64 Opteron2 55.0       2  7857M      -  Yes  () (os64 fta)
ll-fta1 LINUX64 Opteron2 55.0       2  7857M      -  Yes  () (os64 fta)
ll-fta2 LINUX64 Opteron2 55.0       2  7857M      -  Yes  () (os64 fta)
ll-fta3 LINUX64 Opteron2 55.0       2  7857M      -  Yes  () (os64 fta)
ll-fta4 LINUX64 Opteron2 55.0       2  7857M      -  Yes  () (os64 fta)
ll-fta5 LINUX64 Opteron2 55.0       2  7857M      -  Yes  () (os64 fta)
ll-fta6 LINUX64 Opteron2 55.0       2  7857M      -  Yes  () (os64 fta)
ll-fta7 LINUX64 Opteron2 55.0       2  7857M      -  Yes  () (os64 fta)
lc-1        BPROC Opteron2 55.0     -      -      -   No  () (mem8 os32)
lc-2        BPROC Opteron2 55.0     -      -      -   No  () (mem8 os32)
lc-3        BPROC Opteron2 55.0     -      -      -   No  () (mem8 os32)
lc-4        BPROC Opteron2 55.0     -      -      -   No  () (mem8 os32)
lc-5        BPROC Opteron2 55.0     -      -      -   No  () (mem8 os32)
lc-6        BPROC Opteron2 55.0     -      -      -   No  () (mem8 os32)
lc-64      BPROC4 Opteron2 55.0     -      -      -   No  () (mem8 os64)
```



Sample Output From "bhosts -w" on Lightning, October, 2005

```
ll-1%  bhosts -w
HOST_NAME   STATUS       JL/U   MAX  NJOBS   RUN  SSUSP  USUSP  RSV
ll-1        ok              -   510    502   502      0      0    0
ll-2        ok              -   510    124   124      0      0    0
ll-3        closed_Full     -   510    510   484      0      0    0
ll-4        ok              -   510      2     2      0      0    0
ll-5        closed_Full     -   510    510   256      0      0    0
ll-6        ok              -   172      0     0      0      0    0
ll-fta0     ok              -     2      0     0      0      0    0
ll-lsf      closed_adm      -     1      0     0      0      0    0
```

Sample Output From "bhosts -w" on Pink

```
pfe1%  bhosts -w
HOST_NAME    STATUS   JL/U MAX   NJOBS  RUN SSUSP USUSP   RSV
pink         ok       -    1916   1647  871     0     0   376
```

- On Pink and TLC there is only one LSF execution host containing all the nodes in either of those machines. Also, the pink front end pfe2 is NOT an LSF submit host.

- Jobs are allocated entire nodes on all LANL BProc systems, regardless of the actual number of CPUs requested.

- When you are given an allocation of CPUs/nodes through LSF on LANL BProc systems two environment variables associated with BProc are set: **`NODES`** and **`NODELIST`**. See the above example.

  - **`NODES`** specifies which nodes the LSF job can use.
  - **`NODELIST`** lists the processors on each node that the LSF job can use

  LSF has recently been changed so that **`$NODES`** is also echoed directly in your LSF output.

- As with other LANL systems, use **`bkill`** to kill jobs. However, on BProc systems jobs are sometimes reluctant to die. Use these steps (in order):

  1. **`bkill`** ***`job_id`***. If unsuccessful, try

  2. **`bkill -r`** ***`job_id`***. If unsuccessful, try

  3. **`bjobs -l`** ***`job_id`*** **`| grep PGID`** and then **`kill -TERM`** ***`PGID`***. If unsuccessful, try

  4. **`bjobs -l`** ***`job_id`*** **`| grep PGID`** and then **`kill -KILL`** ***`PGID`***. If unsuccessful, try

  5. **`mail consult@lanl.gov`**. Don't forget to mention which machine and which jobid.

Selecting an LSF Execution Host

- On Lightning and Flash there are now machine groups and LSF resources defined that allow you to select different portions of these clusters.

- On Lightning and Flash use the **`-m`** ***`segment_name`*** option to **`bsub`** or **`llogin`**, where *segment_name* is flasha, flashb, or flashc on Flash and ***ll-1, ll-2, ... ll-6*** on Lightning.

- On Lightning the following resources are also defined. Select them with the "**`-R`** ***`resource`***" option to **`bsub`** or **`llogin`**.

Lightning Cluster LSF Resource Definitions	
mem4:	4-GB memory per node;
mem8:	8-GB memory per node;
os32:	32-bit LINUX;
os64:	64-bit LINUX;
fta:	File Transfer Agent;
s_core:	single-core Opteron processor;
d_core:	double-core Opteron processor
ll	Uses an ***ll*** (Lightning only) host
lb	Uses an ***lb*** (Bolt only) host

- Remember that queue structure may prohibit certain combinations of these resources.

LSF Queues

- This section is provided for illustrative use only. Configurations will undoubtedly change as usage grows. You should log in and use **`bqueues -l`** to see current info.

- The LSF queue structure is rather different amongst the BProc machines and is also different in some ways than other LANL systems. Note the following:
    - Fairshare is enabled (queue-based) on all BProc machines and static shares represent project allocations set by the program offices.
    - On Pink there is a new nightq. Nightq accepts jobs anytime, but it will only consider starting jobs from 8:00 pm until 8:00 am and will only run a job if it can complete by 8:00 am. While this queue is active, devq may not have processors available for llogins.
    - Flash has 4 general-access queues: largeq and longq for 32-bit production jobs, debugq for 32-bit development and debugging, and flash64q for 64-bit jobs.
    - Lightning has 3 general-access queues: devq, largeq & smallq. Other queues are for special groups of users, identified by the MMC.

The `bpsh` Command

- The **bpsh** command is used to run a *sequential* command on a slave node.

  The general syntax is:

  ```
  bpsh [bpsh options] node_# command [command args]
  ```

  node_# is the node you were allocated by LSF expressed either as an actual numeric value or as the symbolic representation **$NODES**.

  Note: Use **bpsh** for sequential commands only. If you give **bpsh** a nodelist containing more than one node, it will run copies of the same command on each node.

  Examples (note: all of these assume you've used LSF first, to obtain an allocation of slave nodes):

`bpsh 68 a.out`	Run a.out on node 68
`bpsh $NODES mcnp input=inp output=outp`	An example using the environment variable; typical usage for single-processor runs; (assume $NODES contains a single node)
`bpsh  -p $NODES hostname`	Another example using the environment variable, which, here, can contain multiple values; -p prefixes each line of output with node identifier
`bpsh -p 1,2,3,4  hostname > output`	Execute the **hostname** command on nodes 1-4

- If you try to **bpsh** something on a slave node on which you don't have an allocation you will get an error message:

  Error Message From bpsh Without an LSF Allocation

  ```
  ll-1%  bpsh 1 ls /tmp/hjw
  1: Operation not permitted

  ll-1%  bjobs
  No unfinished jobs found.
  ```

  It also goes without saying that you cannot **bpsh** from a front end!

- Remember:
    - **bpsh** doesn't run a remote shell on the slave node. Commands expecting shell interpretation may fail. Wildcards are expanded by/on the master node, not on the slave node(s). Any shell interpretation of a **bpsh** command always takes place on the master node.

- DO NOT use **bpsh** with the **give** command.

- DO NOT execute things such as perl, csh, etc., using **bpsh**. Run your scripts on the front ends and have the scripts use **bpsh** only for applications. There is a Perl interface to the BProc library, though. Type **man BProc** to find out about it.

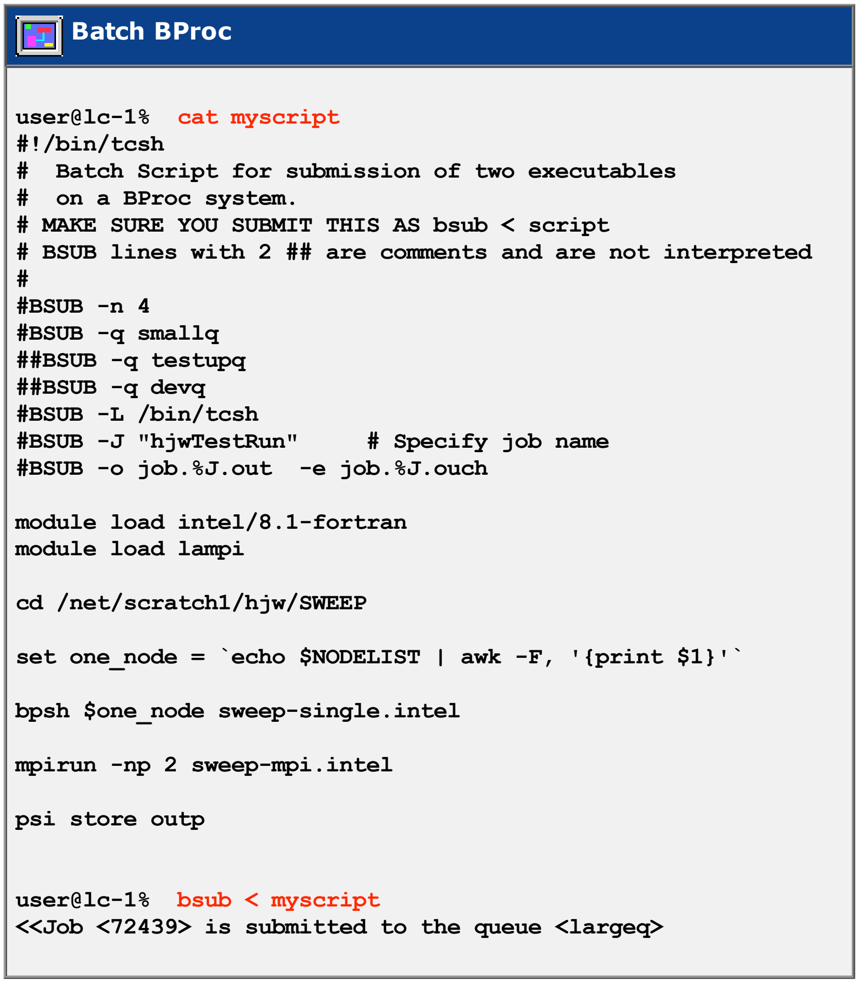
- DO NOT run the **module** command on a slave node using **bpsh**. This will not work. Both interactively and in a batch script run the **module** utility on the front end, and the environment that is set up as a result will be transferred to the slave node(s) when you execute your a.out.

- The following example uses a batch script file. THIS IS AN EXAMPLE ONLY; IT MAY NOT WORK ON ALL SYSTEMS. The things to note are:

  - Do NOT use **bpsh** with **module load**

  - Do NOT use **bpsh** with **cd**

  - Do NOT use #BSUB with **bpsh**



Batch BProc

```
user@lc-1%  cat myscript
#!/bin/tcsh
#  Batch Script for submission of two executables
#  on a BProc system.
# MAKE SURE YOU SUBMIT THIS AS bsub < script
# BSUB lines with 2 ## are comments and are not interpreted
#
#BSUB -n 4
#BSUB -q smallq
##BSUB -q testupq
##BSUB -q devq
#BSUB -L /bin/tcsh
#BSUB -J "hjwTestRun"     # Specify job name
#BSUB -o job.%J.out  -e job.%J.ouch

module load intel/8.1-fortran
module load lampi

cd /net/scratch1/hjw/SWEEP

set one_node = `echo $NODELIST | awk -F, '{print $1}'`

bpsh $one_node sweep-single.intel

mpirun -np 2 sweep-mpi.intel

psi store outp


user@lc-1%  bsub < myscript
<<Job <72439> is submitted to the queue <largeq>
```

Using MPI on BProc Clusters

- To run an MPI job on a BProc cluster you use **mpirun** as the job launcher. Do not use **bpsh** for this because **mpirun** is already "BProc-aware."

- However, you must supply the **-n** or the **-np** argument to **mpirun** even if you supply this argument to **bsub**. (Not the case with Blue Mountain, for example.)

  Note: This will no longer be the case starting with LAMPI version 13.

- Two key things to be careful of: It is possible to execute **mpirun** on a front end (all machines) or master node (Lightning only) without using LSF first; however, your job will execute entirely on the front end or master. You don't want to do this! (Other systems prohibit it.)

It also is possible to *accidentally* run an MPI job requesting more processors than you were allocated from LSF (using `-np`); however, this will oversubscribe the processors that you were allocated.

- As mentioned before, you must load an MPI modulefile in order to run.
- Do NOT "hardwire" the path to mpirun in your scripts. Let the modulefile load handle this.
- There is an extra "administrative" MPI process created per node when you execute an MPI job using LAMPI on all BProc clusters. This process is mostly in the "S" (suspended or sleep) state and doesn't accumulate very much (or any) CPU time.
- Also, beware of differences in LAMPI installation directories (`/opt` vs. `/usr` in latest versions).
- Additional note: If you insist on using MPICH make sure you run with `mpirun -np # --nper 2`. Failure to include this additional parameter will yield the error message `"Not enough nodes to allocate all processes"`. It is recommended that you switch to LAMPI, too.

Exercise

Exercise #4: Using LSF and bpsh

1. First, make sure that you've logged in to a front end using `ssh`.
2. Issue the LSF `llogin` command to get an allocation of one node.
3. After LSF dispatches your `llogin` job which machine are you on? Are you on the front end, the master, or a slave node now?
4. Determine which slave node were you allocated by LSF. There are at least 2 ways to do this.

   Write the answer here: ____________________
5. **Pink or TLC Users Only:** Issue the following commands and explain the result you see:
   1. `cd`
   2. `pwd`
   3. Take the result of that last command and run
      `bpsh $NODES ls result_of_last_command`
6. Issue the following two commands and explain the result you see. Use some unique file name; in other words, substitute your moniker.

   ```
   bpsh $NODES date > /tmp/moniker_file
   ```

   ```
   bpsh $NODES cat /tmp/moniker_file
   ```
7. Now exit from your llogin session.
8. Store a file to HPSS using the hpssq. Do it two ways: interactively and from a batch script or from a batch command line submission.

This ends the fourth exercise.

Contents

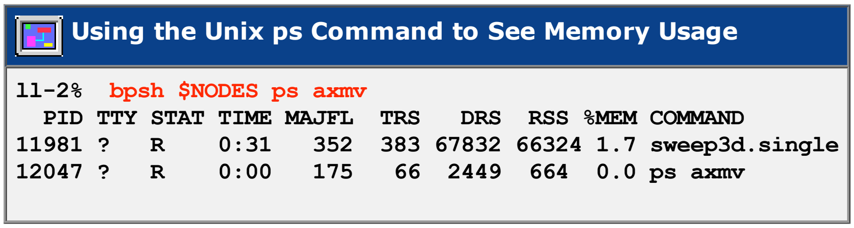
Monitoring Job/System Status on LANL BProc Systems

- On LANL BProc systems there are essentially 3 ways of finding out what's happening on the system:
    1. The Unix way;
    2. the LSF way;
    3. and the BProc way.
- Differences between these 3 ways:
    - The Unix and BProc methods only work on a BProc master node; they do not work from the front ends.
    - On Lightning and Flash, the BProc and Unix methods give status information only for the particular segment on which they are run.
    - In contrast, the LSF commands provide cross-segment status, meaning you can run them from front end and/or master node and get status information for the entire cluster.

1. **Unix commands**: `top` and `ps`. They are segment-specific, meaning if you run one of these commands on ***ll-2*** you'll only get information about ***ll-2***.
    - The Unix `ps` command, if run on a master node, will show all processes, master node and slave node. Slave-node processes are listed with [square brackets].

      See the ps man pages for list of options; typical ones are **`-aux`** and/or **`-efl`**.

      Important use of `ps`: **`bpsh $NODES ps axmv`** will show you how much memory your code is using as it runs on a slave node.

      Using the Unix ps Command to See Memory Usage

      ```
      ll-2%  bpsh $NODES ps axmv
        PID TTY STAT TIME MAJFL  TRS   DRS   RSS %MEM COMMAND
      11981 ?   R    0:31   352  383 67832 66324 1.7 sweep3d.single
      12047 ?   R    0:00   175   66  2449   664 0.0 ps axmv
      ```

    - (An alternate method of observing memory behavior is available here. The software is in /usr/projects/ups/PROCMON/v-02-02-01/ on qsc, flash, qa, qb, and lightning.)
    - The Unix `top` command provides an ongoing look at processor activity in real time. Note that unlike `ps`, if your run `top` on the master node, slave node processes ARE NOT displayed with square brackets. So there's no way to distinguish. However, see **`bptop`**, below.

      After you type `top` you can type **`u`** and give a user id to restrict the top display to that user.
2. **LSF commands**: `bjobs`. A cross-segment command with same syntax as on other LANL systems.
3. **BProc commands**: **`bpstat`**, **`bpps`**, and **`bptop`**. These must be run on a master node to work; if you run them on the front end they give really strange error messages on some of the systems (and basically nothing on the others). On Lightning they all give segment-specific information.
    - `bptop` is a special BProc version of the Unix `top` command. Use this to interactively display either BProc or non-BProc processes in a "top-like" fashion. Use "c" to toggle back and forth between either BProc or non-BProc processes.
    - The **`bpstat`** command shows the status and owner of the slave nodes.

`up`	node is up and available
`down`	node is down or can't be contacted by master
`boot`	node is coming up (running node_up)
`error`	an error occurred while the node was booting

- Make sure you understand how to relate the **bpstat** and **bjobs -u all** output in the following example.

Sample bpstat and bjobs Output

```
ll-2%  bpstat
Node(s)                     Status       Mode    User  Group
0,2,12,13                   down      ---------- root  root
251                         error     ---x------ root  root
1,4-11,14-53                up        ---x------ gre   desktop
27-34                       up        ---x------ jnu   desktop
56                          up        ---x------ scb   desktop
57-58                       up        ---x------ hjw   desktop
3, 59-255                   up        ---x------ root  root

ll-2%  bjobs -u all
JOBID USER STAT QUEUE FROM_HOST EXEC_HOST JOB_NAME SUBMIT_TIME
6503  gap  RUN  devq  ll-1      4*ll-1    /tcsh -l Mar 8 15:11
6482  gre  RUN  devq  ll-1      98*ll-2   viz_job  Mar 8 13:29
6488  jnu  RUN  devq  ll-2      16*ll-2   llogin   Mar 8 14:16
6489  scb  RUN  devq  ll-3      2*ll-2    llogin   Mar 8 14:18
6503  hjw  RUN  devq  ll-1      4*ll-2    /tcsh -l Mar 8 15:11
6503  hjw  RUN  devq  ll-1      16*ll-3   /tcsh -l Mar 8 15:11
```

- Whereas **bjobs -u all** shows info for the entire Lightning cluster (all 6 segments) **bpstat** shows only info local to the BProc master node on which it is run. BProc operations only access a single Lightning cluster segment.

- Nice **bpstat** options: **bpstat -t allup** tells how many nodes are up. Other options include alldown, allunavailable, etc.

- The bpps command is a special version of bpstat; it is basically **bpstat** with **ps**-formatted output.

  bpps will return info about any user job running on the slave nodes (not just your own) but it will not return anything unless a user job is running (and llogin does not count, since it doesn't run on the slave nodes).

bpps	Returns info about all jobs on the cluster segment.
bpps -u userid	Return info about a specific user.
bpps -k	List processes sorted by node.
bpps -s	Ignore processes in the S state.

Using the bpps Command

```
ll-2%  bpps
NODE  USER    PID    PGID   S  STIME      TIME    COMMAND
3     hjw    24774  24747  R  10:36   00:03:02  [sweep3d.mpi]
3     hjw    24775  24747  R  10:37   00:02:12  [sweep3d.mpi]
3     hjw    24784  24792  R  10:38   00:01:26  [sweep3d.mpi]
1     lpm    9069   29044  R   9:59   00:42:06  [MPIping]
1     lpm    9091   29044  S   9:59   00:42:06  [MPIping]
```

- IMPORTANT use of **bpps**: It turns out that, similar to other systems, jobs that terminate in certain ways don't always clean up after themselves. Sometimes this can happen when you interrupt a job using CNTRL-C. Anyway, it's a good idea to use **bpps** to see if any "zombie" processes remain.

Exercise

Exercise #5: Monitoring and Running Jobs

Studying the Machine Status

1. Use the LSF **`bjobs -u all`** command to find all jobs on the system. You may have to pipe this to **`more`**. Or **`less`**.

2. Pick one or two jobs that are in the "RUN" state from the previous step, and figure out which nodes these jobs are using.
   Write the answer here:
   user:__________node(s):___________
   user:__________node(s):___________

3. How many nodes does **`bpstat`** show as down?
   Answer: ________________________
   How many nodes does **`bpstat`** show as unavailable?
   Answer: ________________________
   How many nodes does **`bpstat`** show as up but not allocated to anyone?
   Answer: ________________________

4. Determine if anyone is running a job on the master node or on a front end that shouldn't be run there. There are several ways to do this. Make sure you try at least one way.

   If you find such a job, feel free to send a polite but firm e-mail to that user. Explain to him/her:

   > - Everything that you've learned about BProc;
   >
   > - Why it's important not to run big jobs on the master node;
   >
   > - How to tell the difference between processes running on the master and processes running on the slave.
   >
   > You might want to dictate this letter to yourself regardless of what you find.

   This ends exercise 5.

Running Codes

1. Now you want to execute two applications on the slave node systems. One will be a single-processor application, the other will be an MPI application to be run on two (2) processors.

   You will need two windows on the class machine to do this exercise.

2. The instructor has used the **`give`** utility to give you a file called "**class.tar**." Get the file and copy it to a filespace that you own. Do you remember where a file goes when someone uses **`give`**?

3. Type the command "**tar xvf class.tar**" to extract all the files; then do "**ls**." You should see the following files:

   ```
   sweep-single
   sweep-mpi
   sweep-single.f
   sweep-mpi.f
   timers.c
   input
   testsize.c
   testsize.f
   ```

4. Run the sequential binary (**sweep-single**) so that it runs on a slave node.

   Note: You need to load the compiler modulefile for Intel version 8.1 first.

   Do this any way you choose - interactively or batch. Verify WHILE IT IS RUNNING that it is running on a slave node. Do this with a Unix command or a BProc command IN ANOTHER WINDOW. The application normally sends its output to the terminal; you'll know it completed when it shows a "Wall grind time."

5. Run the MPI binary (**sweep-mpi**) so that it runs on 2 slave-node processors. The output looks basically the same - you should look for the "Wall grind time." Verify, again, WHILE IT

IS RUNNING, that it is running on the slave nodes.

This ends the fifth exercise.

Additional Monitoring Tool

- An additional method of studying machine status: Use the ICN Monitoring Web Site http://icnn.lanl.gov/drm/alljobs. Try it!

Contents

HPSS

- The High-Performance Storage System (HPSS) is used to archive your valued data, large or small, for long periods of time in a safe and secure environment.
- HPSS is available in the secure and yellow networks. It is not available in the Turquoise network.
- You should use the LANL psi command to access HPSS. You can use it on the front ends and the master nodes but it is not available on the compute nodes. There is NO HPSS access from the compute nodes.

File Transfer Agents on Lightning, Flash, and Saguaro

- The best way to do HPSS transfers on these systems is to use the FTAs.
- On Lightning the LSF queue to access this hardware is **hpssq**. On Flash it is called **ftaq**.
- Using hpssq you can run jobs such as these:

  Batch: **`bsub -q hpssq [other bsub options] psi [psi options] psi_command`**

  Interactive: **`bsub -q hpssq -Ip [other bsub options] psi`**

- See http://computing.lanl.gov/article/478.html for complete details. There is also a presentation from the SUF that you can download there.
- On Lightning, using hpssq will cause HPSS transfers to run on the FTAs. On Flash, using hpssq (in the same way as above) will cause HPSS transfers to run on the front ends, at least until the FTA hardware is installed.
- The reason to use hpssq is to reduce computation and network traffic on the front-end and master nodes (Lightning), to use the XPSI interface (Lightning), and because neither PSI nor XPSI is available on the slave nodes (Lightning and Flash).

Contents

More Hardware Details

Opteron Processor (Lightning and Flash)

- The AMD Opteron microprocessor in Lightning and Flash features an Instruction Set Architecture compatible with (and built upon) the Intel x86 architecture. A schematic of the architecture appears below. You can see a photograph of the die here.
- Opteron hypothetical floating-point throughput:
  Double-precision (64-bit): 1 add + 1 mult = 2.2 GFlop/s
  Single-precision (32-bit): 2 add + 2 mult = 4.4 GFlop/s
- Opteron Memory Hierarchy For Floating-Point Data:

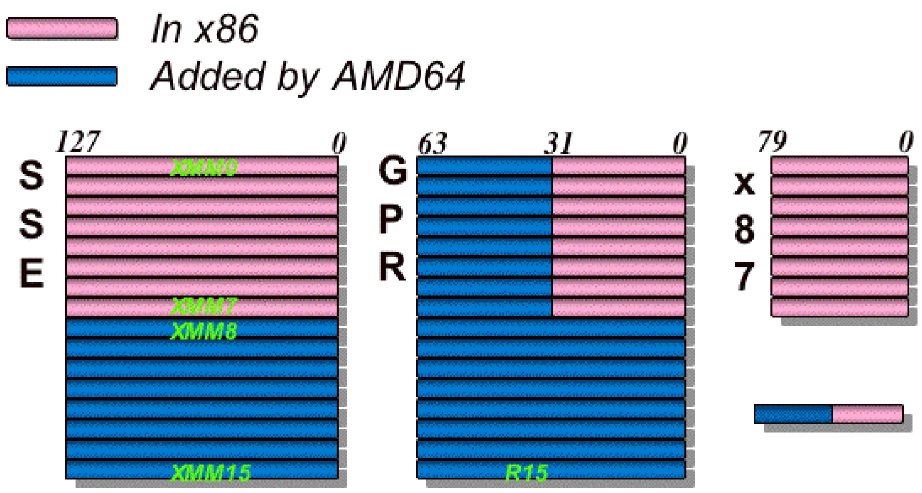
Level	Capacity	Access Time (Clock Periods)
Registers	varies (see below)	1

L1 Cache	64 KB	3
L2 Cache	1 MB	12
Main Memory CPU0	varies	~134
Main Memory CPU1	varies	~206*

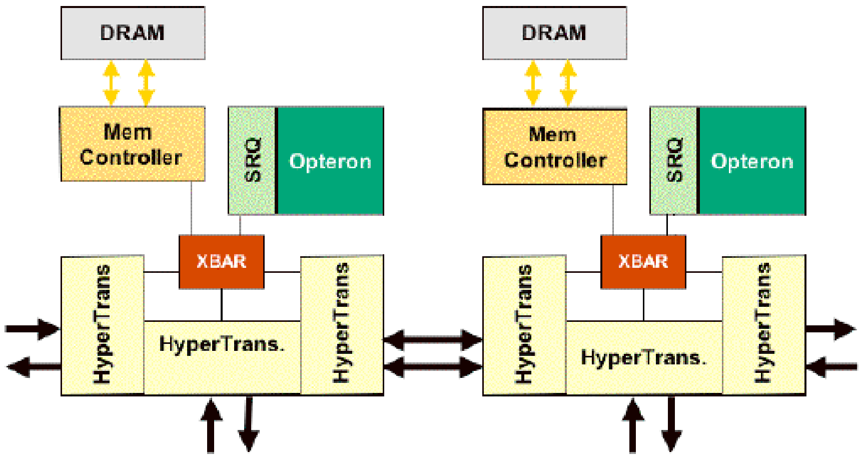
*Data from "Lightning: A Performance and Scalability Report on the use of 1020 nodes," by Kei Davis Adolfy Hoisie Greg Johnson Darren J. Kerbyson Mike Lang Scott Pakin Fabrizio Petrini, LANL CCS-3, http://www.c3.lanl.gov

- Opteron includes an integrated (on-chip) memory controller, intended to reduce DRAM memory latency and increase memory bandwidth. It eliminates the need for a front-side bus and runs at the processor speed, (not at the bus speed, as it did in older processors).
- A new technology called HyperTransport is used as the on-chip interconnect interface. Each Opteron has 3 HyperTransport data links (two for communication between processors, one for the rest of the system). Each HyperTransport link has a peak transfer bandwidth of 6.4GB/s.

AMD Instruction Set Architecture



2-Processor AMD Opteron™ Server



- The dual HyperTransport links for CPU data are coherent links that share the memory and cache space between the two processors in a node. Each of these processors has one-half of the node's memory. This means two things:

1. A single **`malloc`** can only access one-half the total memory per node (2 GB or actually,

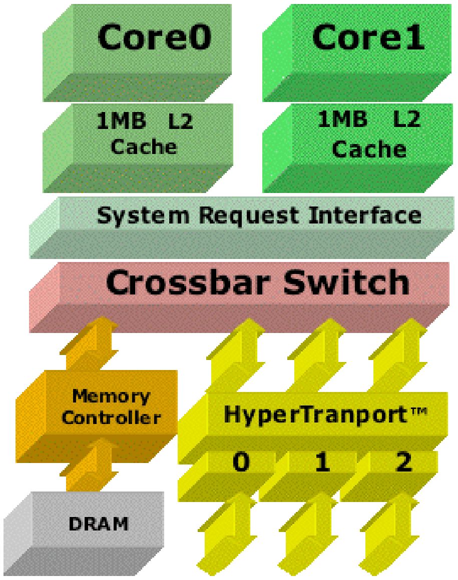
about 1.75 GB after the OS grabs its share); and

2. When a processor accesses memory from the other processor in a node there can be a 25% - 50% increase in memory latency.

- A single node of the Lightning cluster physically looks like this. The compute nodes are mounted five to a "sub-chassis" (pictured here. and there are four sub-chasses per standard 19"-deep cabinet.

- The Opteron processor is also being used in the Sandia National Laboratory Red Storm (that's a PDF file) supercomputer (provided by Cray, Inc.) and other supercomputers from Cray.

- Newer versions of the Opteron are available in a dual-core architecture and one segment of Lightning will use these. A schematic of the architecture is shown in the figure below. The key points are that each core has own separate L1/L2 cache hierarchy but the cores share the Integrated Memory Controller and HyperTransport links.



Xeon Processor (Pink)

- Will not be covered here. See http://asci-training.lanl.gov/Pink/index.html#HardwareArch and references therein.

Myrinet

- The System Area Network (SAN) in Lightning, Flash, and Pink is a cost-effective, high-performance, packet-communication and switching technology called Myrinet, from Myricom, Inc.

- Myrinet consists of four basic parts: a host interface that connects to the computing nodes' PCI I/O bus, a low-latency, high-bandwidth communication switch, Myrinet fiber-optic cables, and some low-level Myrinet message-passing software known as GM.

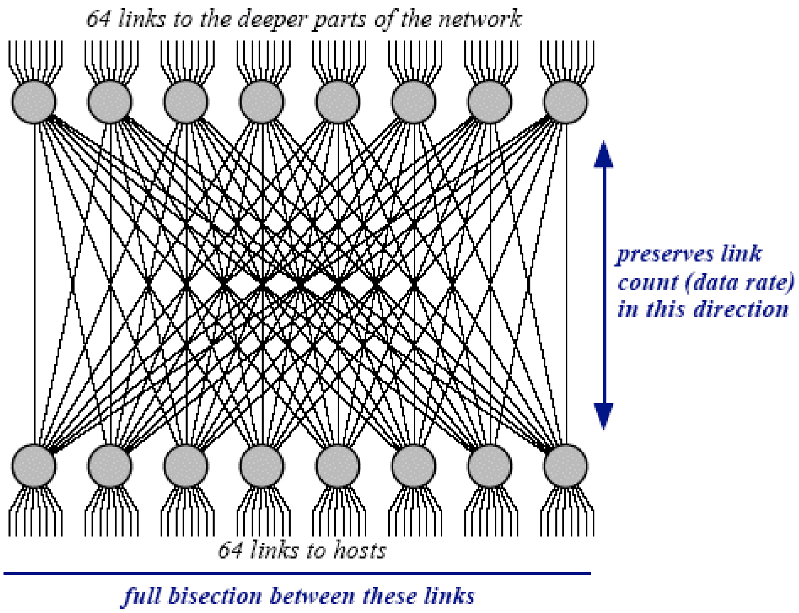
- The basic building block of the Myrinet network is a single-chip 16-port crossbar switch. Cards containing this switch are combined into a single enclosure supporting up to 128 hosts. Federated networks are created combining these 128-host enclosures, organized as either "leaf/line" switches (connecting to NICs) or "spine" switches (connecting other switches).

- There is one "rail" of interconnect. Measurements on Myrinet have shown an MPI round-trip latency of 6-7 microseconds and a peak transfer bandwidth of about 250 MB/s per link.

- Myrinet uses a Clos network topology, a multi-stage, non-blocking network, similar to a fat-tree (bandwidth scales as log # of processors).

- Lightning has a single network for the entire cluster that could scale to 2,048 hosts. There are two levels of line switches and a top level set of spine switches. Worst-case transmission involves 7 switch hops.

- A schematic of a smaller Clos network is here . Each of the 16 sub-networks at the bottom layer of this diagram is actually a full-bisection 64-host network:


64 links to the deeper parts of the network
preserves link count (data rate) in this direction
64 links to hosts
full bisection between these links

- Myrinet is a source-routed network. i.e., each host must know the route to all other hosts through the switching fabric.
  - For the current version of Myrinet there's a Mapper program running as a user process on each node that automatically discovers all of the hosts connected to the Myrinet network, computes a set of deadlock-free minimum-length routes between the hosts, and distributes appropriate routes to each host on the connected network. (The next version is supposed to have it running as a kernel thread.)
  - On Lightning we use a mapper written by Erik Hendriks, of CCS-1. You can read a paper about it (PDF format).

Contents

Porting Considerations

- Perhaps the most important issue, one that takes a bit of getting used to, is that there are no shells on the slave nodes. All shell scripts and shell commands must be run on the front ends.
- Other key issues to be aware of include: data format, if transferring data from Q or C* systems, 64-bit IEEE floating point, 64-bit pointers, system calls, and OpenMP support.

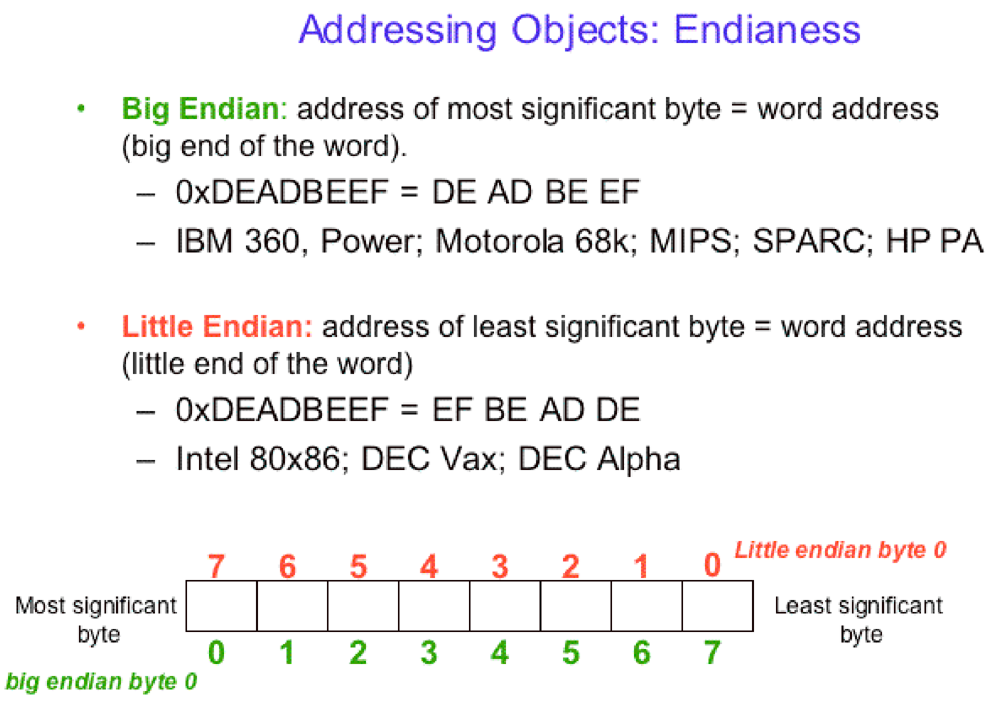
Big Endian vs. Little Endian

- This refers to the order in which the machine stores the bits of a word. It is a characteristic of the processor. The following table shows byte order for important systems at LANL:

Byte Order of Some Important Systems

Machine	Processor	Byte order
ASC Q	DEC/Compaq/HP Alpha EV68	Little Endian
Lambda, Pink, Grendels, Lightning, Flash, Coyote, Saguaro	Intel Pentium	Little Endian
Blue Mountain	MIPS R10000	Big Endian

Mauve	Intel Itanium	Littl Endian
ASC White	IBM Power	Big Endian



Addressing Objects: Endianess

- **Big Endian:** address of most significant byte = word address (big end of the word).
  - 0xDEADBEEF = DE AD BE EF
  - IBM 360, Power; Motorola 68k; MIPS; SPARC; HP PA
- **Little Endian:** address of least significant byte = word address (little end of the word)
  - 0xDEADBEEF = EF BE AD DE
  - Intel 80x86; DEC Vax; DEC Alpha

- Two basic Endian-related issues:
  - Code or compilation issues
    - Dereferencing pointers on Blue Mountain vs. Q. Thanks to John Daly for providing this example. Consider the following C program:

```
#include main()
{
        long *a;
        int *b;
        a = (long *) malloc(8);
        a[0] = 0x1234567890abcdef;
        b    = (int *)a;
        printf("A: %lx, B: %lx %lx\n", a[0], b[0], b[1]);
}
```

On Blue Mountain this program yields:

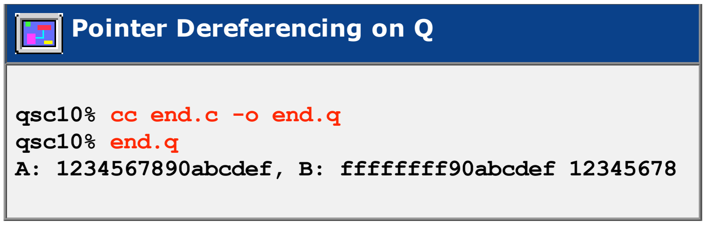
Pointer Dereferencing on Blue Mountain

```
t01% cc -64 -o end.theta.64 end.c
cc-1178 cc: WARNING File = end.c, Line = 9
  Argument is incompatible with the corresponding format
string conversion.

          printf("A: %lx, B: %lx %lx\n", a[0], b[0], b[1]);
cc-1178 cc: WARNING File = end.c, Line = 9
  Argument is incompatible with the corresponding format
string conversion.

          printf("A: %lx, B: %lx %lx\n", a[0], b[0], b[1]);
t01% end.theta.64
A: 1234567890abcdef, B: 12345678 ffffffff90abcde
```

On Q this program yields:



Pointer Dereferencing on Q

```
qsc10% cc end.c -o end.q
qsc10% end.q
A: 1234567890abcdef, B: ffffffff90abcdef 12345678
```

- Runtime issues
  - Data files written out on Blue Mountain or Theta must be converted to little-Endian format before they can be read on Q, Lightning, Flash, and Pink systems.
  - Let's say you have a data file that you created on machine Blue Mountain that you want to read in to a Fortran program on Lightning. Some of the Fortran compilers on Lightning have compile-line options that convert the file on the fly.
  - To Convert a data file from big- to little-Endian in C you have to write your own routine to reverse the bit order. Or you can use the one the consultants have here.

Compiling

CompileNodes

- Lightning, Flash, and Pink clusters have special nodes used for compiling. You don't compile on the BProc master or slave nodes. The compile nodes are the same as the front ends.
  - **This means you don't `llogin` before compiling**, which is different from other LANL systems.
  - For example, on Pink, compile on pfe1 and/or pfe2. The compilers WILL NOT work on the BProc master node, pink.lanl.gov.
  - There are no slave nodes associated with the compile node(s). After you log in to the compile node (with `ssh`), simply `module load` whatever modulefiles you need, and start compiling. No BProc commands are needed to build your code; just use "make" or "configure" or "f95" or whatever.
  - All compile nodes have 2 CPUs. Parallel builds are limited to 2 processors.
- CPP is in `/usr/bin/cpp`
- perl is in `/usr/bin/perl` (NOT where it is on theta or QSC!)

32- Vs. 64-Bit Computing

- The Opteron microprocessor in Lightning, Flash, Saguaro, Coyote, and TLC is a native 64-bit architecture.
- However, we have been using Lightning and Flash in what is called "Legacy Mode," which is designed to be fully compatible with 32-bit Intel systems. In this mode the operating system itself is a 32-bit OS and the compilers have been generating only 32-bit binaries. See the table below.
- This has had four important effects: (1) 32-bit address space (maximum `malloc`=2 GB); (2) maximum file size 2 GB (although see below); (3) a variety of performance-related effects; (4) floating-point computation done in an unsual way.

Mode	OS	Application Re-compile?	Address Size (bits)	Register Extend?	GPR Width (bits)

Long Mode	64-bit Mode	New 64-bit OS	Yes	64	Yes	64
	Compat. Mode		No	32	No	32
Legacy Mode		Legacy 32-bit	No	32	No	32

- Some development work is now proceeding in 64-bit mode and we plan to phase in 64-bit computing on some newer Lightning and Flash segments. You'll need to note the following:

    1. Applications compiled in 32-bit mode can run without recompilation on 64-bit segments, as long as all the libraries used in the 32-bit compilation are available and loaded into your environment (with modulefiles). Apps run this way *should* produce the same numerical results, too. This is called "Long Mode Compatibility Mode" in the table.
    2. The real power of the Opteron is fully exposed in "Long Mode 64-Bit Mode." This requires complete recompilation, potentially using certain compiler switches, and using 64-bit libraries loaded from certain modulefiles. An example compiler switch is **`-tp k8-64`** for PG. LAMPI will have separate modulefiles for 32-bit and 64-bit mode.
    3. When used in Long Mode 64-bit Mode the Opteron allows a flat address space of up to 2^48. That's 282 TB and it ought ot hold most ASC users for a while.
    4. The Opteron microprocessor does floating-point computation in a very different way in 32-bit mode than in 64-bit mode. Hence, numerical results may be different between Legacy mode and 64-bit mode.
    5. On 64-bit systems there is a new compiler, PathScale, that can only be used on 64-bit systems.
    6. You can determine if a given relocatable binary or an executable is 32-bit or 64-bit using the Unix "**`file`**" command.
    7. Note that none of this has anything to do with the precision required by your application. If the numerics of your application require 64-bit floating-point precision you can generally achieve this on virtually any processor through proper data declarations - ensuring that all data objects are represented by programming language data types that contain sufficient storage, i.e., "double precision" in Fortran77 parlance. However, you should get slightly better 64-bit floating-point performance in Opteron's Long Mode 64-bit Mode.

Big File Fix

- A fix allowing files larger than 2 GB in 32-bit Linux systems has been available for some time. You need to apply *all three of* the following when you compile:

    -D_FILE_OFFSET_BITS=64

    -D_LARGEFILE64_SOURCE=1

    -D_LARGEFILE_SOURCE=1

    on the gcc compile line. These preprocessor flags work with gcc and intel C. For the PGI compilers use these and add **`-Mlfs`**. There is no such combination that works for the NAG compiler. You can check by **`'nm -B a.out'`** and checking that things like 'open64' are defined, not plain 'open'.

    Then, if you want to randomly "seek" within a file, after using the above compiler options, it is recommended that you use **`lseek`** in conjunction with **`open`** instead of other combinations, such as **fopen, fseek**, etc.. If this isn't possible, contact the ICN Consultants, who can provide further advice.

Available Compilers

- The following compilers are available, although not on all systems. Each compiler must be accessed by loading its corresponding modulefile. Be aware that the same name may be used by several compilers (i.e., **f95** from both NAG and Absoft). There is a proposal to go forward with only two compilers, PGI and PathScale.
    - GNU
        - Fortran (**g77**), C (**gcc**), and C++ (**g++**) are available. (No f90.)

          One version available without loading a modulefile. Generally, one or two other versions available via modulefiles.
        - Debugging: **-g**
    - Absoft
        - The compilers are **f77, f90, f95** (and cc).
        - Fortran optimization options: **-O0, -O1, -O2, -O3**
        - Other options of interest include:
            - **-YEXT_SFX="_"**: append an underscore suffix to subprogram names.
            - **-YEXT_NAMES=LCS**: subprogram names will have lower case letters.
            - **-i8** promote integers to i*8
            - **-N113** promote REALs and COMPLEX variables to DOUBLE (or DOUBLE COMPLEX).
    - Intel
        - This is the Intel x86 compiler producing code for IA32.
        - There are two significantly different versions available.
          With version 7.1 you load a single modulefile and get both the Intel Fortran compiler **ifc** and the Intel C compiler **icc** .
        - With version 8 you need to load a separate modulefile for Fortran (intel-fortran-8.1) and C (intel-c-8.1).
          With version 8 Fortran is invoked with **ifort**. C is still **icc**.
        - Another difference: with Fortran version 8 you need to have the compiler's modulefile loaded in order to run you code; with version 7, you don't.
        - Another difference: with Fortran version 7 there is a "default" version but with version 8 there is not. Be careful with "**module load intel**"
        - Optimization with -O1 (default), -O0 (none), or -O3
        - Debugging with -g
        - **-r8 -i8**
        - **-auto** (default for scalar vars) or -save
        - **-Vaxlib**: links in some important "portability" routines (**ETIME, GETARG, GETENV, IARGC, SHIFTL,** , etc.). Only needed for version 7.
        - **-fpp2**: Fortran preprocessor
        - **-pc<32|53|80>**: internal FP precision <32-|53-|64-bit> significand; pc80 is default
    - Lahey
        - The compilers are **lf95** and **cc or gcc**
        - Fortran optimization options: **--o2 --sse2**
        - Fortran 64-bit real variables: **--dbl**
        - Debugging: **-g** (and **-O0**)
        - Note: The Lahey compilers will NOT be available when the BProc clusters change to 64-bit mode, so you might want to migrate away from them right now.
    - NAG
        - Two compiler versions available: native AMD64 and x86
        - The compilers are **f95** (that's it - no f77 or f90)
        - ABI Choice: **-abi=32** or **-abi=64**; the latter will only run on systems with 64-bit OS (currently none).
        - Fortran optimization options: **-O4, -O3, -O2 (default, equiv. to -O), -O1, -O0**
        - Other options of interest include:
            - **-float-store** don't store FP vars in registers. (For machines with registers wider than 64 bits.)
            - **-ieee=stop: traps FP overflow, Db0, invalid operand; causes execution termination**
            - **-C=array:** bounds checking
            - **-info:** output compiler info messages (default is not to)

- Portland Group
  - The compilers are **pgf77**, **pgf90**, pgCC, **pgcc** and pghpf.
  - Fortran optimization options: **-fastsse -tp k8-64**
    Caution: this option will only work on systems with 64-bit OS (currently none).
  - Fortran 64-bit real variables: **-r8**
  - Note: a current "issue" relating to the PG compiler modulefiles requires that you load its modulefile last if also using a TotalView modulefile.
  - Debugging: **-g** (compiler sets to **-O0**)
  - Other options of interest include:
    - **-Kieee=strict**: strict conformance with IEEE 754 fp standard.
    - **-i8 -r8**:
    - **-Mlfs**: link in Linux routines for large files (> 2 GB)
    - **-Minform=inform**: display all compiler err messages
    - **-fast**: -O -Munroll -Mnoframe
    - **--c**: array bounds checking
    - **-byteswapio**: Swap bytes from big-endian to little-endian or vice-versa on I/O of unformatted data.
  - NEW: The AMD Core Math Library (ACML), a set of numerical routines tuned specically for Opteron processors, is available. The routines, which are available in both FORTRAN and C interfaces, include BLAS, LAPACK, FFT, and fast random number generators.
    - To use it with PG compilers two steps are required: compile with **-Mcache_align** and link with **-lacml**
    - Documentation is available here.
- PathScale
  - Compilers are **pathf90**, **pathcc**, and **pathCC**. There is no pathf77.
  - Ofast Equivalent to -O3 -ipa -fno-math-errno
    -OPT:roundoff=2:Olimit=0:div_split=ON:alias=typed.

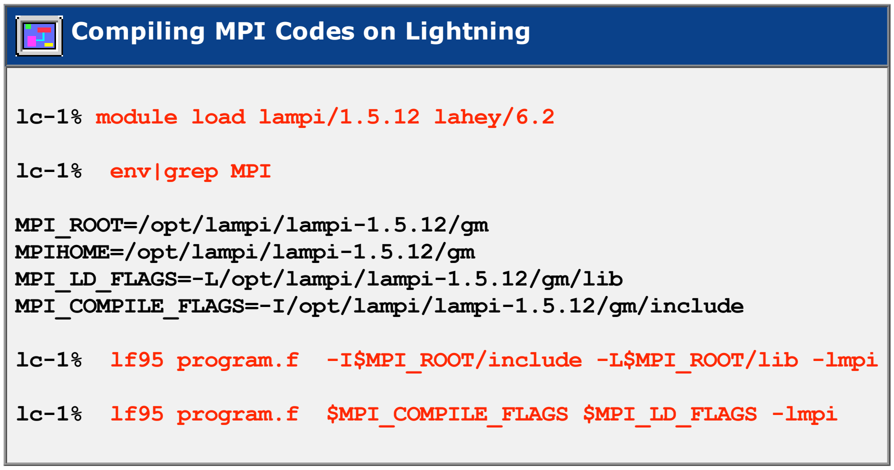
    ipa is interprocedural analysis. Optimizes across functional boundaries. Must be specified both at compile and link time.

    Aggressive unsafe optimizations: Changes order of evaluation. Deviates from IEEE 754 standard to obtain better performance.
  - **-byteswapio** writes all data in format opposite to that of native processor.
  - **-conversion [native, little_endian, big_endian]**
  - There is temporarily an ACML library available. You can link against it with

    -L/net/scratch1/dog/flash64/AMD/acml_3.1.0/pathscale64/lib -lacml

Compiling MPI Codes

- There are two MPI packages available on Lightning and Pink now: LAMPI, the Los Alamos Message Passing Interface and a version of MPICH, a version from Argonne Nat'l Lab. However, we expect that MPICH will go away soon. MPICH is not available on Flash.
- Neither of these packages is supplied/supported by a vendor.
- The Los Alamos Message Passing Interface (LA-MPI) project provides an end-to-end network fault-tolerant message passing system for tera-scale clusters. You can read about the project on http://public.lanl.gov/lampi.
- Of course, you need to load a modulefile in order to use either MPI package.
- On all BProc systems LAMPI is fully compatible with all compilers, implicitly. That means there are not separate modulefiles for LAMPI compiled with one compiler or another.
- The LAMPI modulefiles are of the form **lampi/*version***.
  The MPICH modulefiles are of the form **mpich/*version***.
- As is the case with other LANL systems (such as QSC) your compile lines need to include the path to the MPI include files and load libraries. Example:



Compiling MPI Codes on Lightning

```
lc-1% module load lampi/1.5.12 lahey/6.2

lc-1%  env|grep MPI

MPI_ROOT=/opt/lampi/lampi-1.5.12/gm
MPIHOME=/opt/lampi/lampi-1.5.12/gm
MPI_LD_FLAGS=-L/opt/lampi/lampi-1.5.12/gm/lib
MPI_COMPILE_FLAGS=-I/opt/lampi/lampi-1.5.12/gm/include

lc-1%  lf95 program.f  -I$MPI_ROOT/include -L$MPI_ROOT/lib -lmpi

lc-1%  lf95 program.f  $MPI_COMPILE_FLAGS $MPI_LD_FLAGS -lmpi
```

- Note that the environment variables set by loading the MPI modulefile on the BProc clusters are different from the ones set by loading the MPI modulefile on the Q clusters. You may have to change your makefile.

LAMPI Tips

1. Diagnostic options:

   `mpirun -t` Tags output from parallel processes with a prefix indicating its origin, e.g. a prefix of 9[4.1.n16] means process rank 9, host 4, on-host process rank 1, host name n16

   `mpirun -w` Enables library warnings (about requesting 0 bytes of memory, temporary resource exhaustion etc.)

   `mpirun -v` Verbose library diagnostics, primarily related to initialization and termination.

2. Performance Options

   `mpirun -f` Disables argument checking, slightly lowering latency.

   `mpirun -mf noack, nochecksum` This disables LA-MPI guarantee of the integrity of transmitted data using a checksum/retransmission protocol, which would incur a small but non-negligible overhead on communication. This is a reasonable default mode of operation if the network is regarded as stable.

3. Other options

   `mpirun -q` Disables the LA-MPI startup banner.

   `mpirun -crc` Enables 32-bit CRC checking instead of 32-bit additive checksums (the LA-MPI default) for application to application data integrity.

4. Configuration File

   - Default options can be set in a configuration file **`~/.lampi.conf`**. Each command line option has an associated configuration file variable that can be set in this file. The complete set of options and corresponding configuration variables can be listed using "**`mpirun -list-options`**".

   - For example, to enable diagnostic tagging of output, turn off argument checking, the start-up banner and Myrinet checksum/retransmission, put the following lines in **~/.lampi.conf**

```
# Tag output, don't check args, no banner and assume Myrinet is reliable
OutputPrefix   1
NoArgCheck     1
Quiet          1
MyrinetFlags   noack,nochecksum
```

5. Log File

   - Error and warning messages from LA-MPI are recorded in a log file called **lampi.log** in the current directory.

     The name of this file can be changed by setting the environment variable **LAMPI_LOG**.

     To disable the log file

     ```
     setenv LAMPI_LOG /dev/null
     ```

6. Processes

   - As mentioned above, LA-MPI creates an administrative daemon process on each slave node that creates and manages the application processes. There is also a master-node process "**mpirun ...**" created. A useful way of filtering out the admin MPI processes is to use the **bpps -s**, which will ignore processes in the "S" state.

Exercise

Exercise #6: Compiling

1. Compile testsize.c or testsize.f and run it on a slave node. Note the results. Here is what you should see.

2. Compile the sweep-single file into a sequential binary executable. You don't have to run it, just compile it. Use your favorite compiler. You need two source files: **sweep-single.f** and **timers.c** . Compile it using the debugging flag (**-g**).

3. Compile the sweep-mpi.f file into an MPI binary executable. You don't have to run it, just compile it. Use your favorite compiler. Or your least favorite. You need two source files: **sweep-mpi.f** and **timers.c** . Compile it using the debugging flag (**-g**).

This ends the sixth exercise.

Contents

Debugging

- There is a Portland Group debugging tool, pgdbg, if you used the PG compilers.

- Thanks to the gallant efforts of Laurie McGavran, Totalview runs on Lightning, Flash, Pink, TLC, and Grendels. There are different procedures for debugging serial and parallel jobs.

- At the time of this writing there are several versions of Totalview available on the BProc systems. The examples below do not mean to imply that any one version is necessary or preferred.

- You will always be running TotalView on the master nodes of the BProc clusters.

- Note: see the compiler section above for a special note on using Totalview with PG compilers.

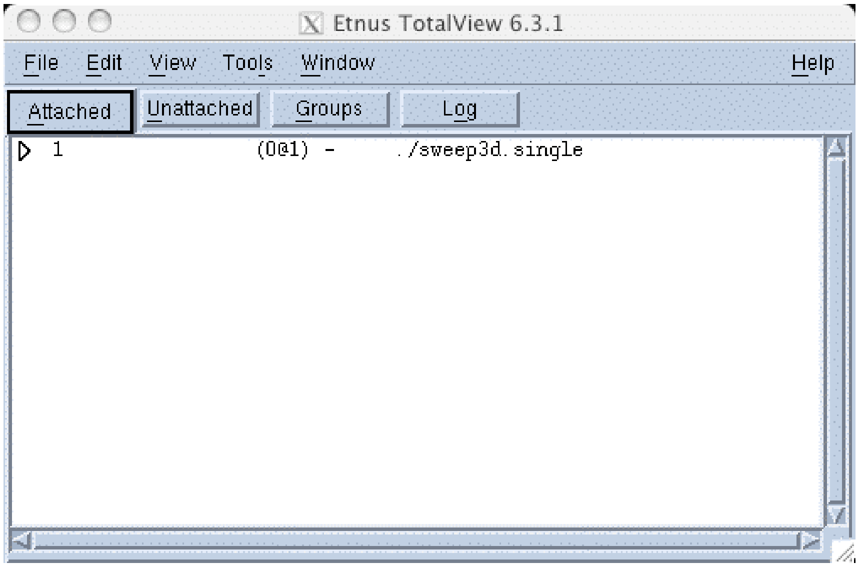
- Additional Note: If you're using Pink see the Xauth section above for a special note on setting up X windows, which is needed for TotalView.

Debugging a Serial Job With TotalView

- In this case, you run a TotalView server remotely on a slave node that talks to a GUI running on the front end. Here's how to use TotalView (on the master node) to debug a serial job running on a remote slave node.

```
llogin
module load totalview/version
totalview -remote $NODES ./a.out
```

- When you do this it will bring up the TotalView "root window" (shown below for an executable called "sweep3d.single."):
- To start debugging, double click on the line containing the executable name. This will bring up the TotalView "process window" and you can then debug as usual. When you start the executable it will be running on the slave node.



```
1          (0@1) -     ./sweep3d.single
```

Debugging a LAMPI MPI Job With TotalView

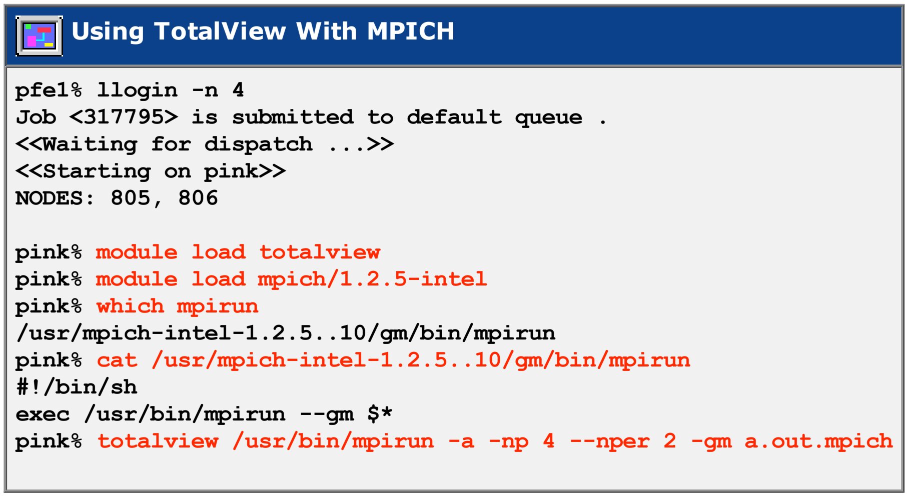
- The general procedure is as follows:

```
llogin -n #
module load totalview/version lampi
totalview mpirun -a -np # ./a.out
```

- The behavior should be the same as on other LANL systems: First, type "Shift-G" in the Process Window; a "Question" Window will appear asking if you want to stop the job" and you should answer "Yes" if you want to set breakpoints.

NEW!Debugging an MPICH Job With TotalView NEW!

- A special procedure is needed for this because **mpirun** for MPICH isn't an executable, it's a script. The following procedure works as of May, 2006. Remember that there are several builds of MPICH on Pink, one for each compiler. The example below assumes that the application code was built with the Intel version of MPICH. You can see that not all of these steps are required - some extra ones are included to explain what's going on.



Using TotalView With MPICH

```
pfe1% llogin -n 4
Job <317795> is submitted to default queue .
<<Waiting for dispatch ...>>
<<Starting on pink>>
NODES: 805, 806

pink% module load totalview
pink% module load mpich/1.2.5-intel
pink% which mpirun
/usr/mpich-intel-1.2.5..10/gm/bin/mpirun
pink% cat /usr/mpich-intel-1.2.5..10/gm/bin/mpirun
#!/bin/sh
exec /usr/bin/mpirun --gm $*
pink% totalview /usr/bin/mpirun -a -np 4 --nper 2 -gm a.out.mpich
```

Contents

Exercise

Exercise #7 (Optional): Debugging Running Codes On Slave Nodes

1. This exercise is optional and considers an advanced feature of TotalView not covered in the tutorial. Do this only if you already know how to use TotalView well.

2. **Attach to a running (slave-node) sequential binary.**

    1. Start sweep-single running on a slave node.

    2. In another window **get on the master node and `cd` to where you compiled sweep in the previous exercise.** Then start TotalView with no arguments.

    3. In TotalVIew's Root Window click on the "Unattached" button to bring up a pane showing all processes owned by you but not attached to TotalView.

    4. In TotalVIew's Root Window select File -> New Program from the menu bar. Enter the name of the executable with its full path, its PID, and the number of the slave node on which it is running. (If you get an error message about a missing shared library, quit TotalView, load the modulefile for the compiler that was used to build sweep-single, and restart.)

    5. (Note that you cannot attach to a remote process by diving on it in the Unattached Pane, because you can only do this with local processes, but here, you're running TotalView on the master node and the process on a slave node.)

    6. After you've attached, "Halt" the process to see the source code in TotalView's Process Window. You can "Go" the process and/or then quit TotalView.

3. **Attach to a running (slave-node) MPI-parallel binary.**

    1. Start sweep-**mpi** running on a slave node **with 2 processes**.

    2. In another window start TotalView with no arguments.

    3. In TotalVIew's Root Window click on the "Unattached" button to bring up a pane showing all processes owned by you but not attached to TotalView.

    4. Dive on the "mpirun" process. This should give you control of your MPI processes.

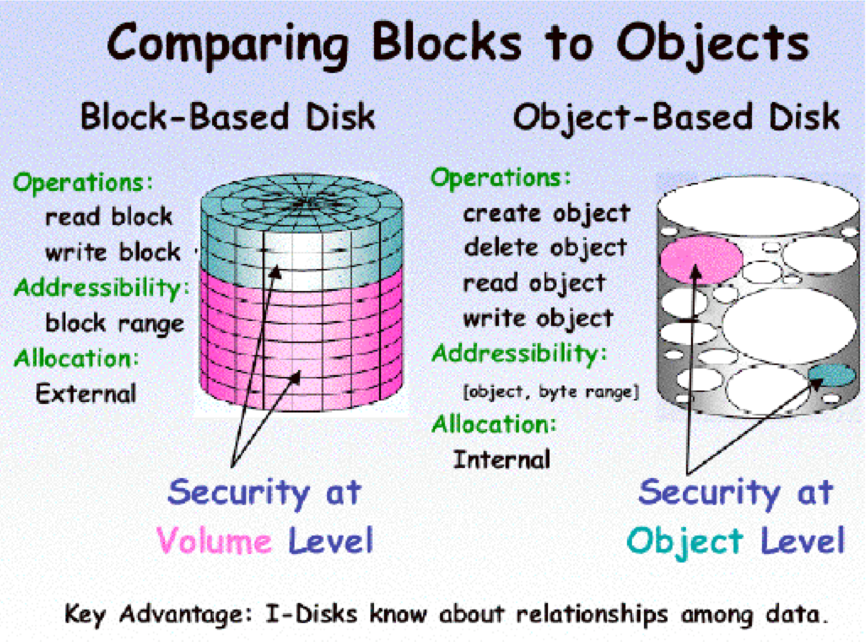
    5. To see the source code, go to the "Attached" pane in the Root Window. You may have to "expand" the mpirun processes (but not the first one). After you expand, dive on one.

This ends the optional debugging exercise.

Panasas File System

- The global storage and parallel filesystem for all BProc clusters is provided by Panasas, a company founded and currently led by Garth Gibson, who did the research that led to the development of RAID technology.

- The Panasas product is an example of an important new storage technology called Object Based Storage, in which the storage devices have some "understanding" of how different blocks of a file are related. Block-to-file mapping and inode processing is offloaded to the storage devices.



- Another key aspect is separation of data and metadata amongst different storage devices and datapaths. In Panasas metadata is stored on "Director Blades" and user data objects are stored on "Storage Blades."

- Files larger than 64 KB are striped across the Panasas storage devices. This is done to improve performance; i.e., the user application can potentially realize the aggregate throughput of multiple storage devices. Rebalancing of the system is done automatically and is object based.

- Other OBD activities include lustre.org and IBM's StorageTank initiative. Lustre has been deployed successfully at LLNL.

- To the user, the Panasas system should appear as a highly-available, global shared file system with relatively high performance, even for small, sequential file I/O but especially for large-scale parallel I/O.

- Initial performance: expect 40 MB/sec/Storage Blade, thus, 400 MB/sec maximum for a file striped across one shelf, 10 storage blades. 300 MB/sec is routine. From an individual machine (node) expect 70-90 MB/sec. Metadata rates (inserts/deletes/lookups/stats per second) run at roughly NFS rates. Parallel operations scale to multi-GB/s levels.

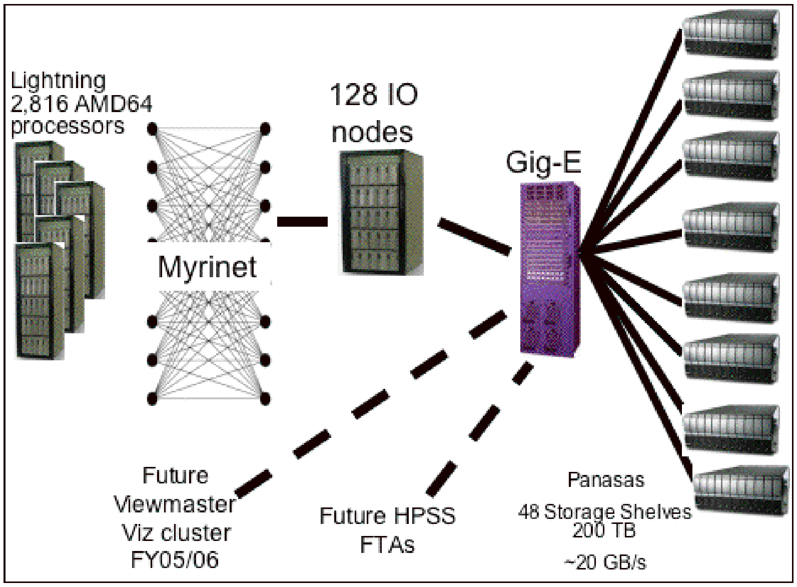
- If you use MPI-IO or are interested in maximizing performance of your application's I/O performance on Panasas you should view the presentation given by James Nunez recently. It's a PowerPoint file available on http://computing.lanl.gov/article/439.html.

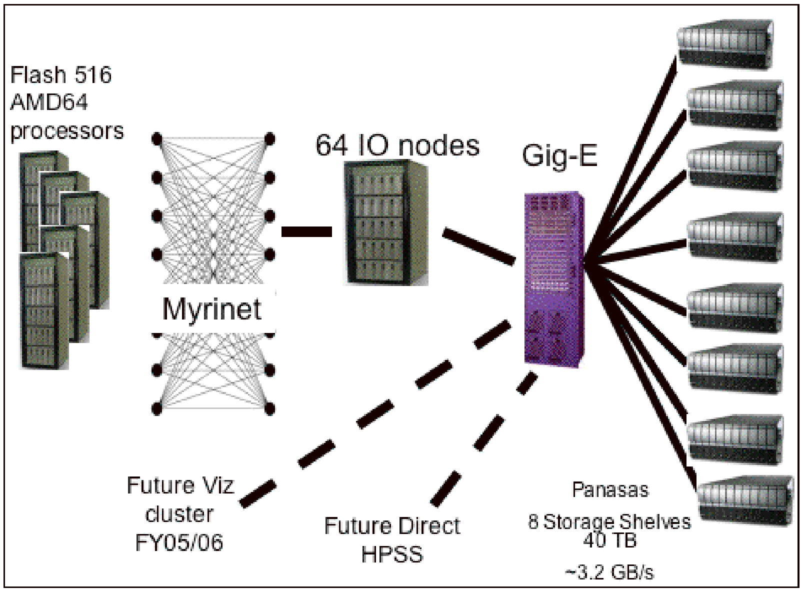
- On all LANL BProc systems Panasas is connected to the cluster via a set of GigE switches. All I/O requests go over the Myrinet interconnect. Several nodes serve as I/O nodes - essentially Myrinet-to-GigE routers.

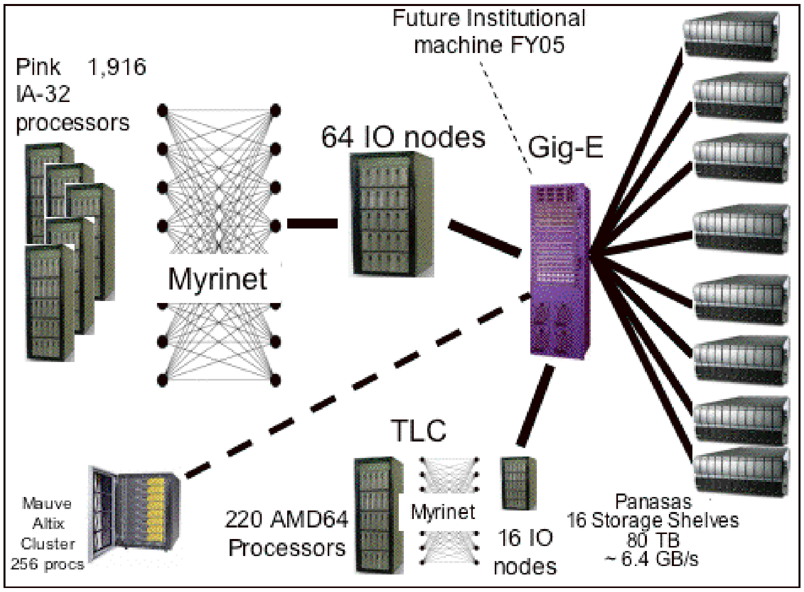
- In a Panasas system 10 Storage Blades and 1 Director Blade are combined into a single shelf containing 5 TB of storage and a 16-port Gigabit Ethernet switch (which uses 4 ports to the network and 11 to the blades). The blades use Intel Pentium processors and 250-GB ATA disk drives.

11-blade shelf
H:7" [4U] x W: 19"
5 TB

- The Panasas system on Lightning, Flash, Pink, and TLC will be mounted on the front ends, master nodes, and slave nodes as **`/net/scratch1`** and **`/net/scratch2`**.
- As mentioned above, on Pink, the PanFS filesystem is the only filesystem that the slave nodes have access to.






Future Institutional machine FY05
Pink 1,916 IA-32 processors
64 IO nodes
Gig-E
Myrinet
TLC
Mauve Altix Cluster 256 procs
220 AMD64 Processors
Myrinet
16 IO nodes
Panasas 16 Storage Shelves 80 TB ~ 6.4 GB/s

Contents

Where to Go For Help

- LANL's ICN Consulting Office will handle questions on Lightning. You can reach them at 5-4444, option "3", or send email to consult@lanl.gov

- Please note the recent declaration of support levels for LANL systems, available on http://computing.lanl.gov/article/454.html.
- There is Lightning, Flash, and Pink information on the HPC Documentation page, http://computing.lanl.gov in both the open and the secure.

Contents

Future Improvements

- 64-bit Linux 2.6, Bproc V4 (Milestone O), Posix threads, OpenMPI, PaScalBB (Scalable and available I/O network design), HPSS FTAs
- More nodes/segments (bolt, flashd).
- Increase Panasas on Lightning to 200TB.

Contents

Reference Info

- Flash Quick Reference Guide
- Lightning Quick Reference Guide
- Pink-TLC Quick Reference Guide
- Saguaro Quick Reference Guide (coming soon).

- Coyote Quick Reference Guide (coming soon).
- The BProc Project Software Repository http://sourceforge.net/projects/bproc
- BProc Project Description http://bproc.sourceforge.net
- LANL's Cluster Research Team http://public.lanl.gov/cluster
- Clustermatic Home Page http://www.clustermatic.org
- Original page on LANL's Pink System http://www.lanl.gov/projects/pink
- Excellent description of Intel x86 architecture by Patterson & Hennessy.
- Linux Bios Home Page http://www.linuxbios.org
- "BProc: The Beowulf Distributed Process Space", Erik A. Hendriks, 16th Annual ACM International Conference on Supercomputing , June 22-26 2002. Available as PS or PDF on http://public.lanl.gov/cluster/papers/index.html
- A news article about lightning from fcw.com
- Linux Networx's Home Page http://www.linuxnetworx.com
- Linux Networx's take on LinuxBios linuxnetworx.com/products/linuxbios.php
- Scyld Computing Corp., http://www.scyld.com/platform_overview.html
- Linux Labs
- The Advanced Computing Laboratory's Ed cluster (postscript file), an early BProc prototype.

Contents

Course Evaluation

Evaluation Form

Please complete the online evaluation form at http://trouble.lanl.gov/~hjw/eval.php in the yellow network. Click the box above to go there.

Los Alamos
NATIONAL LABORATORY
EST.1943

Public Web Site | Web Contact | Privacy Policy |
Operated by the University of California for the U.S. Department of Energy |
LA-UR-05-2741.